Static strategies for worksharing with unrecoverable interruptions

Anne Benoit, Yves Robert, Arnold Rosenberg, Frédéric Vivien

Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Static Strategies for Worksharing with Unrecoverable Interruptions

Anne Benoit · Yves Robert · Arnold L. Rosenberg · Frédéric Vivien

the date of receipt and acceptance should be inserted later

Abstract One has a large workload that is "divisible"—its constituent work's granularity can be adjusted arbitrarily—and one has access to p remote *worker* computers that can assist in computing the workload. How can one best utilize the workers? Complicating this question is the fact that each worker is subject to interruptions (of known likelihood) that kill all work in progress on it. One wishes to orchestrate sharing the workload with the workers in a way that maximizes the expected amount of work completed. Strategies are presented for achieving this goal, by balancing the desire to checkpoint often—thereby decreasing the amount of vulnerable work at any point—vs. the desire to avoid the context-switching required to checkpoint. Schedules must also temper the desire to replicate work, because such replication diminishes the effective remote workforce. The current study demonstrates the accessibility of strategies that provably maximize the expected amount of work when there is only one worker (the case $p = 1$) and, at least in an asymptotic sense, when there are two workers (the case $p = 2$); but the study strongly suggests the intractability of exact maximization for $p \geq 2$ computers, as work replication on multiple workers joins checkpointing as a vehicle for decreasing the impact of work-killing interruptions. We respond to that challenge by developing efficient heuristics that employ both checkpointing and work replication as mechanisms for decreasing the impact of work-killing interruptions. The quality of these

A portion of this research appeared at the *23rd IEEE International Parallel and Distributed Processing Symposium [6].*

A. Benoit, Y. Robert
École normale supérieure de Lyon and the Université de Lyon
and Institut Universitaire de France, France.

A. Rosenberg
Colorado State University and Northeastern University, USA.

F. Vivien
École normale supérieure de Lyon and the Université de Lyon and INRIA, France.

heuristics, in expected amount of work completed, is assessed through exhaustive simulations that use both idealized models and actual trace data.

Keywords Fault-aware scheduling, Unrecoverable interruptions, Divisible load, Probabilistic scheduling.

1 Introduction

Technological advances and economic constraints have engendered a variety of modern computing platforms that allow a person who has a massive, compute-intensive workload to enlist the help of others' computers in executing the workload. The resulting cooperating computers may belong to a nearby or remote cluster (of "workstations"; cf. [24]), or they could be geographically dispersed computers that are available under one of the increasingly many modalities of Internet-based computing—such as Grid computing (cf. [11, 13, 14]), global computing (cf. [12]), or volunteer computing (cf. [20]). In order to avoid unintended connotations concerning the organization of the workers, we avoid evocative terms such as "cluster" or "grid" in favor of the generic "assemblage."

Advances in computing power never come without cost. The new "collaborative" platforms add various types of *uncertainty* to the list of concerns that must be addressed as one prepares a computational workload for allocation to the available computers. When one allocates work over the Internet, for instance, one must prepare for computers to produce results much more slowly than anticipated, even, possibly, failing ever to complete their allocated work. The current paper follows in the footsteps of sources such as [3,5,10,17,22,28], which present analytic studies of algorithmic techniques for coping with uncertainty in computational settings. Whereas most of these sources address the uncertainty of the computers in an assemblage one computer at a time, we view the assemblage here as a "team" wherein one computer's shortcomings can be compensated for by other computers, most notably if we judiciously *replicate work*, i.e., allocate some work to more than one computer.

The problem we study. We have a large computational workload whose constituent work is *divisible*, in the sense that one can subdivide chunks of work into arbitrary granularities (cf. [9]). We have access to $p \geq 1$ identical *worker* computers to help us compute via *worksharing*, a modality of collaborative computing in which the owner of the workload allocates work to remote computers that are available; cf. [29]. In the current paper we study only *homogeneous* assemblages of workers, in order to concentrate only on the problem of coping with uncertainty within an assemblage. In companion work, we have begun to focus on the added complexity of coping with uncertainty within a *heterogeneous* assemblage [8].

We focus here on the most draconian type of uncertainty that can plague an assemblage of computers, namely, vulnerability to *unrecoverable interruptions* that kill all work currently in progress on the interrupted computer. We strive

to cope with such interruptions—whether they arise from hardware failures or from a loaned/rented computer's being reclaimed by its owner, as during an episode of *cycle-stealing* (cf. [3,10,26,27,28]). The two tools that we employ to cope with these interruptions are *checkpointing*, which saves already completed work so that it will not be killed by an interruption, and *work replication*, which allocates some work to more than one worker. The only extrinsic resource to help us use these tools judiciously is our assumed *a priori* knowledge of the risk of a computer's having been interrupted—which we assume is the same for all computers.[1]

The goal of our study. We strive to maximize the *expected amount of work that gets computed by the assemblage of computers*, no matter which, or how many, computers get interrupted. We call a schedule that achieves this goal *optimal*. Thus, we posit that within our application even partial output is meaningful. The annotation of metagenomics data is a timely such application: failing to classify all available DNA fragments (as *eukaryotes, prokaryotes*, etc.) just artificially augments the "unknown" category.

The challenges. The challenges of scheduling a workload on interruptible workers can be described in terms of two dilemmas. *(1)* Sending work to workers in small chunks lessens vulnerability to interruption-induced losses, but it also increases the impact of per-work-unit overhead and minimizes the opportunities for "parallelism" within the assemblage of workers. *(2)* Replicating work lessens vulnerability to interruption-induced losses, but it also minimizes the expected productivity advantage from having access to many workers. (The pros and cons of work replication are discussed at length in [19].)

Approaches to the challenges. *(1) "Chunking" our workload.* We strive to balance overhead costs with the risk of interruptions, thereby coping with the first dilemma, by allocating work to each worker as a sequence of carefully sized *chunks*.[2] This approach, which is advocated in [10,26,27,28], allows each worker to *checkpoint* according to some schedule, thereby protecting its work from the threat of work-killing interruptions. *(2) Replicating work.* We strive to give all chunks of work a high likelihood of being computed successfully, despite the second dilemma, by using the (known) probability of interruptions to choose when and where to allocate some chunks to more than one worker.

Because the costs of checkpointing and of communicating with workers are typically high in time and overhead, we limit such activities by orchestrating work allocation (and replication) in an *a priori,* static manner, rather than dynamically, in response to observed interruptions. While we thereby duplicate work unnecessarily when there are few interruptions among the workers, we thereby prevent *our* computer, which is the server in the studied scenario, from becoming a bottleneck when there are many interruptions.

[1] As in [10,26,28], our scheduling strategies can be adapted to use statistical, rather than exact, knowledge of the risk of interruption—albeit at the cost of weakened performance guarantees.

[2] We use the generic "chunk" instead of "task" to emphasize tasks' divisibility (which precludes *atomic* tasks).

Summary. We assume: that we know the instantaneous probability that a worker will have been interrupted by time t; that this probability is the same for all workers and that it increases with the amount of time that a computer has been available (whether working or not). These assumptions, which we share with [10, 26, 28], seem to be necessary in order to derive scheduling strategies that *provably* maximize the expected amount of work completed. Our analytical results illustrate how quickly our target problem tends toward analytical intractability as the setting get closer to "reality": by adding workers (Section 4) and/or allowing workers to differ in power ([8]).

Main results. Our early work on this topic is reported somewhat sketchily in the conference report [6]. The current paper fleshes out and extends the (analytical) results from that report that deal with one and two workers; we similarly expand on the portion of [6] that deals (experimentally) with arbitrary numbers of workers, and we extend our heuristics to handle *arbitrary* interruption-risk models.

Section 3 treats the case of *a single worker*. We describe the one-computer schedules from [6] that are *exactly* optimal in work production for the interruption model in which risk increases *linearly* with time, both for scenarios that assess per-chunk overheads and those that do not. We end with a schedule that is *asymptotically* optimal in work production under *arbitrary* interruption-risk models. We turn in Section 4 to extending the study in [6] of worksharing with *two workers*. The difficulty of this extension forces us to focus only on scenarios that do *not* assess per-chunk overheads—our *free-initiation model*; however, one can convert our free-initiation results to scenarios that *do* assess per-chunk overheads, that are predictably close to optimal in work production (see Theorem 1 in Section 2.2.4). The difficulty of the two-worker case also forces us, in certain situations, to settle for schedules that are *asymptotically* optimal. Our main results provide the following insights into the two-worker case.

- Guidelines for optimal scheduling in any scenario wherein interruption risk never decreases with time (Theorem 5).
- A family of schedules $\{\Sigma(n)\}_{n\geq 1}$ whose quality, under arbitrary risk, tends to optimal as the number n of deployed chunks grows without bound (Theorem 6). (We term this *asymptotic optimality*.)
- Exactly optimal "symmetric" scheduling under linear risk when work is deployed in a single chunk (Theorem 7). ("Symmetric" is defined in the Theorem.)
- Asymptotically optimal scheduling under linear risk with multi-chunk deployment (Algorithm 1 and Theorem 8).

After studying the two-worker case, we turn to two complementary studies of simple, well-structured, heuristics for sharing work with arbitrary numbers of workers. (1) In Section 5, we develop strategies for crafting such heuristics; and, employing differing intuitions about how to complete a lot of work in expectation no matter what the risk of interruption, we craft six specific heuristics. The schedules produced by the *greedy* heuristic are found to domi-

nate the other heuristics' schedules in expected work production, albeit at the cost of a bit more computation. (2) In Section 6, we extend our study of the linear risk function by comparing the schedules produced by the best of the preceding six heuristics against those produced by four simple heuristics. We determine via simulations which heuristics perform better as we vary workload sizes, number of workers, checkpointing granularity, checkpointing overhead, and degree of work replication. As in Section 5, the *greedy* heuristic dominates the others. In Section 7, we go beyond the linear risk model by adapting our new heuristics for use with arbitrary risk functions, and we evaluate the adapted heuristics using actual traces.

Related work. The literature contains relatively few rigorously analyzed scheduling algorithms for interruptible "parallel" computing in assemblages of computers. Among those we know of, only [3,10,26,27,28] deal with an *adversarial* model of interruptible computing. One finds in [3] a randomized scheduling strategy which, with high probability, completes within a logarithmic factor of the optimal fraction of the initial workload. In [10, 26, 27, 28], the scheduling problem is viewed as a game against a malicious adversary who seeks to interrupt each worker in order to kill all work in progress. (Also, [6] is a preliminary version of the current paper.) Among the experimental sources, [31] studies the use of task replication on a heterogeneous desktop grid whose constituent computers may become definitively unavailable; the objective is to eventually process all work. In a similar context, [1] aims at minimizing both the completion time of applications and the amount of resources used. There is a large literature on scheduling divisible workloads on assemblages of computers that are not vulnerable to interruptions. We refer the reader to [9] and its myriad intellectual progeny; another good start is [4]—a thorough study of divisible load scheduling on star and tree networks—or [16]. Also on the subject of divisibl workloads, it is shown in [15, 16] how a linear model, such as our free-initiation model, can lead to absurd schedules involving infinitely many infinitely small chunks; we cope with this issue in Section 3.1.1. We do not enumerate here the many studies of computation on assemblages of workers, which focus either on systems that enable such computation or on specific algorithmic applications. However, we point to [21] and [30] as exemplars of the two types of studies.

2 The Technical Framework

We now convert the informal discussion in the Introduction into a framework for developing and rigorously validating scheduling guidelines.

2.1 The Computation and the Computers

We have $W_{(\mathrm{ttl})}$ units of divisible work to execute on an assemblage of $p \geq 1$ identical *worker* computers. Each worker is vulnerable to *unrecoverable* interruptions that "kill" all work in progress on it. All workers share the same

instantaneous probability of being interrupted, which we know exactly. This probability increases with the amount of time the worker has been operating—whether it has been computing or not.

As discussed in the Introduction, the danger of losing work in progress when an interruption occurs mandates that we not just divide our workload into $W_{(\mathrm{ttl})}/p$ equal-size allocations and deploy one chunk on each worker. Instead, we "protect" our workload as best we can, by:

- partitioning it into *chunks* of possibly different sizes; "chunk" is our term for a unit of work that we allocate to a worker;
- prescribing a schedule for allocating chunks to workers;
- allocating some chunks to more than one worker, as a divisible-load analogue of work replication.

As noted earlier, we treat intercomputer communication as a resource to be used very sparingly. Specifically, we avoid having *our* computer become a communication bottleneck by orchestrating chunk replication in an *a priori,* static manner—rather than dynamically, in response to observed interruptions.

2.2 Modeling Interruptions and Expected Work

2.2.1 The interruption model

Within our study, all workers share the same *risk function*, i.e., the same instantaneous probability, $Pr(w)$, of having been interrupted by the end of "the first w time units." We measure time via work units that *could be* completed "successfully" if there is no interruption. In other words, "the first w time units" is the period of time that a worker would need to complete w units of work *if* it started working on them when the entire worksharing episode begins, *and* it succeeded in completing them. This time scale, which is shared by all workers, thus uses the start of the worksharing episode as the baseline moment for measuring time and risk. Recall that $Pr(w)$, which we assume that we know *exactly*, cannot decrease as w increases.

It is useful to generalize the preceding measure of risk by allowing many baseline moments. We denote by $Pr(s, w)$ the probability that a worker *has not been* interrupted during the first s "time units" but *has been* interrupted by "time" $s+w$. Thus, $Pr(w) = Pr(0, w)$, and $Pr(s, w) = Pr(s+w) - Pr(s)$. We let[3] $\kappa \in (0, 1]$ be a constant that weights our probabilities; we illustrate the role of κ as we introduce the risk function used in most of our results.

Linearly increasing risk. Most of our results focus on the *linear risk function* $Pr(w) = \kappa w$. It is the most natural model of risk in the absence of additional information: the risk of a worker's being interrupted grows linearly with the time that it has been available, or equivalently, to the amount of work it *could have* done. The relevant probability density function is then

[3] As usual, we use parentheses to denote *open* boundaries for real intervals and brackets to denote *closed* boundaries.

$\mathrm{d}Pr = \kappa \mathrm{d}t$ when $t \in [0, 1/\kappa]$ and 0 otherwise, so that

$$Pr(s, w) = \min\left\{1, \int_s^{s+w} \kappa \mathrm{d}t\right\} = \min\{1, \kappa w\}.$$

The constant $1/\kappa$ recurs often in our analyses, since it can be viewed as the time by which an interruption is certain (with probability 1) to have occurred. To enhance legibility of the rather complicated expressions that populate our analyses, we denote the quantity $1/\kappa$ by X.

The following scenario suggests why the linear risk model is relevant to cycle-stealing episodes. On Friday evening, a PhD student has a large set of simulations to run. She has access to the computers in her lab, but each computer can be reclaimed at any instant by its owner. In any case, everybody will be back to work on Monday 8am. What is the student's best strategy? How much simulation data should she send to, and execute on, each available computer?

2.2.2 Expected work production

We use risk functions to help us efficiently maximize the expected amount of work completed by the assemblage, by chunking work for, and allocating work to, the workers. Let $\mathsf{W}_{(\mathrm{cmp})}$ be the random variable whose value is the number of work units that the assemblage completes successfully under a given schedule. Our goal is to maximize the expected value of $\mathsf{W}_{(\mathrm{cmp})}$.

We study how to schedule large workloads under two cost models which differ in how they assess chunk execution-times as functions of chunk sizes. One observes two classes of costs incurred during each communication or checkpointing: the costs that are proportional to the size of a chunk and those that are fixed constants. When chunks are very large, the fixed costs are negligible compared to the size-proportional ones—so one can safely ignore the fixed costs. But one must be careful! If one ignores the fixed costs, then there is never a disincentive to deploying the workload in[4] $n+1$ chunks rather than n; in fact, this increases the expected work production! However, increasing the number of chunks tends to make chunks smaller—which increases the significance of the fixed costs! We deal with this dilemma by recognizing two cost models and striving for optimal schedules under each.

1. The *free-initiation model* accounts for only the size-proportional costs of worksharing. Because it focuses on situations wherein the fixed costs are negligible compared to the size-proportional ones, it does *not* assess a per-chunk fixed cost.
2. The *charged-initiation model* accounts for both the fixed and the size-proportional costs of worksharing. It thus reflects in greater detail the costs incurred by real computing systems.

[4] Throughout, n denotes a positive integer.

The *free-initiation* model. Our results under this model approximate reality well when we allocate work in large chunks. This occurs, for instance, when large fixed costs for each communication or checkpointing event lead us to have each worker do a substantial amount of work between successive events, in order to amortize these costs. In such situations, we keep chunks large by placing a bound on the number of scheduling "rounds," in order to counteract this model's tendency to increase the number of "rounds" indefinitely. Importantly also: knowing the expected value of $\mathsf{W}_{(\mathrm{cmp})}$ under the free-initiation model affords us easy bounds on the corresponding expected value under the charged-initiation model (see Theorem 1 in Section 2.2.4). Such bounds are quite valuable whenever the charged-initiation model is prohibitively difficult to analyze directly.

Within the free-initiation model, we denote the expected value of $\mathsf{W}_{(\mathrm{cmp})}$ under a given schedule Σ, with workload $W_{(\mathrm{ttl})}$, by $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma)$, the superscript "f" denoting "free"(-initiation). The value of this expectation is

$$E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma) \;=\; \int_0^\infty Pr(\mathsf{W}_{(\mathrm{cmp})} \geq u \text{ under } \Sigma)\; du.$$

The *charged-initiation* model. This model is much harder to analyze than the free-initiation model, even when there is only one worker. In compensation, this model often allows one to determine analytically the best numbers of chunks and of "rounds," even when there are *multiple* workers. Under this model, the overhead for each additional chunk is a fixed cost, ε work units, that is added to the cost of computing each chunk. The cost ε could represent, e.g., the setup time for a communication or the overhead of a checkpoint.

We denote by $E^{(\mathrm{c})}(W_{(\mathrm{ttl})}, \Sigma)$ the expected value, under this model, of $\mathsf{W}_{(\mathrm{cmp})}$, for a given schedule Σ and workload $W_{(\mathrm{ttl})}$; the superscript "c" denotes "charged" (-initiation). This expectation is

$$E^{(\mathrm{c})}(W_{(\mathrm{ttl})}, \Sigma) \;=\; \int_0^\infty Pr(\mathsf{W}_{(\mathrm{cmp})} \geq u + \varepsilon)\; du.$$

2.2.3 Observing the models on one worker

We now consider how to optimize expected work-production when there is a single worker, under both the free-initiation and charged-initiation models.

The *free-initiation* model. We calculate $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma)$ for an arbitrary risk function Pr, for three cases wherein Σ deploys the entire $W_{(\mathrm{ttl})}$-unit workload on a single worker. To enhance legibility, let the phrase "under Σ" within the expression "$Pr(\mathsf{W}_{(\mathrm{cmp})} \geq u$ under $\Sigma)$" be specified implicitly by context.

When schedule Σ_1 deploys the workload in a *single chunk*, we have

$$E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_1) \;=\; W_{(\mathrm{ttl})}\left(1 - Pr(W_{(\mathrm{ttl})})\right). \tag{1}$$

To deploy the workload in *two chunks,* schedule Σ_2 partitions it into chunks of respective sizes $\omega_1 > 0$ and $\omega_2 > 0$, where $\omega_1 + \omega_2 = W_{(\text{ttl})}$; we then have
$E^{(\text{f})}(W_{(\text{ttl})}, \Sigma_2) = \int_0^{\omega_1} Pr(\mathsf{W}_{(\text{cmp})} \geq u)du + \int_{\omega_1}^{\omega_1+\omega_2} Pr(\mathsf{W}_{(\text{cmp})} \geq u)du$
$= \omega_1(1 - Pr(\omega_1)) + \omega_2(1 - Pr(\omega_1 + \omega_2))$.
To deploy the workload in n *chunks,* schedule Σ_n partitions it into chunks of respective sizes $\omega_1 > 0, \ldots, \omega_n > 0$, where $\omega_1 + \cdots + \omega_n = W_{(\text{ttl})}$; we have

$$E^{(\text{f})}(W_{(\text{ttl})}, \Sigma_n) = \int_0^{\omega_1} Pr(\mathsf{W}_{(\text{cmp})} \geq u)du \ + \cdots + \int_{\omega_1+\cdots+\omega_{n-1}}^{\omega_1+\cdots+\omega_{n-1}+\omega_n} Pr(\mathsf{W}_{(\text{cmp})} \geq u)du$$
$$= \omega_1(1 - Pr(\omega_1)) \ + \cdots + \ \omega_n(1 - Pr(\omega_1 + \cdots + \omega_n)). \quad (2)$$

The *charged-initiation* model. Mimicking the development for the free-initiation model: When the entire workload is deployed as a single chunk, we have $E^{(\text{c})}(W_{(\text{ttl})}, \Sigma_1) = W_{(\text{ttl})}\big(1 - Pr(W_{(\text{ttl})} + \varepsilon)\big)$, and when it is deployed as two chunks of respective sizes ω_1 and ω_2, we have $E^{(\text{c})}(W_{(\text{ttl})}, \Sigma_2) = \omega_1(1 - Pr(\omega_1 + \varepsilon)) + \omega_2(1 - Pr(\omega_1 + \omega_2 + 2\varepsilon))$. In general, $E^{(\text{c})}(W_{(\text{ttl})}, \Sigma_n)$ is the charged-initiation analogue of the free-initiation expectation $E^{(\text{f})}(W_{(\text{ttl})}, \Sigma_n)$.

Optimizing expected work-production. The goal of our study is to craft schedules Σ that maximize $E^{(\text{f})}(W_{(\text{ttl})}, \Sigma)$ or $E^{(\text{c})}(W_{(\text{ttl})}, \Sigma)$. As a first step toward this goal, we observe that under many risk functions—including the linear one—the workers are *certain* to have been interrupted by some known eventual time. In such situations, one often gets more work done in expectation by not deploying the entire workload: making the last-deployed chunk a bit smaller than would be needed to deploy all $W_{(\text{ttl})}$ units of work actually increases the expectation. (One observes this in Theorems 2 [for the free-initiation model] and 4 [for the charged-initiation model].) Thus, when scheduling a single worker:

- select n chunk sizes that sum to *at most* $W_{(\text{ttl})}$, and select n chunks having these respective sizes,
- deploy these chunks in a way that maximizes the expected amount of completed work.

We formalize this goal for the free-initiation model via the function $E_n^{(\text{f})}(W_{(\text{ttl})}) = \max\{\omega_1(1-Pr(\omega_1))+\cdots+\omega_n(1-Pr(\omega_1+\cdots+\omega_n))\}$, where the maximization is over all n-tuples $\{\omega_1 \geq 0, \ \ldots, \ \omega_n \geq 0\}$ such that $\omega_1 + \cdots + \omega_n \leq W_{(\text{ttl})}$.

2.2.4 Relating the models

One can bound the expected work completed under the charged-initiation model via the analogous quantity for the free-initiation model—which is one "excuse" for focusing on the simpler model. The reader can verify the following for the single-worker case ($p = 1$) directly from Eq. (2) and its charged-initiation analogue. The theorem remains valid for multiple workers.

Theorem 1 (Relating the models) *Let $E_n^{(\text{c})}(W_{(\text{ttl})})$ (resp., $E_n^{(\text{f})}(W_{(\text{ttl})})$) denote the optimal n-chunk expected value of $\mathsf{W}_{(\text{cmp})}$ under the* charged-*initiation*

model (resp., the free-*initiation model) under an arbitrary risk function Pr. For any collection of p workers,*

$$E_n^{(\mathrm{f})}(W_{(\mathrm{ttl})}) \;\geq\; E_n^{(\mathrm{c})}(W_{(\mathrm{ttl})}) \;\geq\; E_n^{(\mathrm{f})}(W_{(\mathrm{ttl})}) - n\varepsilon. \tag{3}$$

Proof The lefthand inequality in (3) being obvious, we focus just on the right-hand inequality, which turns out to be subtler than one might expect.

Let Σ be an optimal schedule for p workers and risk function Pr under the free-initiation model. Σ deploys the workload $W_{(\mathrm{dpl})}$ in n chunks, $\mathcal{W}_1, \ldots, \mathcal{W}_n$, of respective sizes $\omega_1 > 0, \ldots, \omega_n > 0$. For $j \in [1,p]$, $\Sigma(j,k)$ denotes the kth chunk executed on worker j under Σ; that is, worker j executes chunks in the order $\Sigma(j,1), \ldots, \Sigma(j,n)$.[5]

Note that distinct chunks may overlap; i.e., we may have $\mathcal{W}_i \cap \mathcal{W}_j \neq \emptyset$ even when $i \neq j$. For the purposes of our anlysis, we therefore *partition* $W_{(\mathrm{dpl})}$ into pieces $\{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ so that, for any $\mathcal{X}_i$ and $\mathcal{W}_j$, either $[\mathcal{X}_i \subseteq \mathcal{W}_j]$ or $[\mathcal{X}_i \cap \mathcal{W}_j = \emptyset]$. Thus, for all $i \in [1,m]$, if Σ deploys any part of $\mathcal{X}_i$ on worker j, then the latter attempts to compute $\mathcal{X}_i$ in its entirety. Let Π_i be the set of workers that $\mathcal{X}_i$ is deployed on. For $j \in \Pi_i$, $\sigma(j,i)$ is the rank of the first chunk scheduled on worker j that contains $\mathcal{X}_i$; formally $\sigma(j,i) = \min\{k | \mathcal{X}_i \subset \Sigma(j,k)\}$.

Define a new schedule Σ' from Σ as follows. For each i in $[1,n]$, let $\omega_i' = \max\{0, \omega_i - \varepsilon\}$, and let $\mathcal{W}_i'$ be any size-ω_i' subset of $\mathcal{W}_i$. Σ' deploys chunks $\mathcal{W}_1', \ldots, \mathcal{W}_n'$ in the *exact* manner that Σ deploys chunks $\mathcal{W}_1, \ldots, \mathcal{W}_n$ on the same workers, except that null chunks are skipped (to avoid the cost ε). We account for these zero-length chunks in the sequel, via the function

$$\mathbf{1}_{\omega_\mathbf{i}'} \;=\; \textbf{if } [\omega_\mathbf{i}' \neq \mathbf{0}] \textbf{ then 1 else 0}$$

We then define $\mathcal{X}_i'$, Π_i' and $\sigma'(j,i)$ as we did $\mathcal{X}_i$, Π_i and $\sigma(j,i)$, except that we now insist that each $\mathcal{X}_i'$ be a subset of some $\mathcal{X}_k$; we thus refine the partition $\mathcal{X}$ to the partition $\mathcal{X}'$. Let $\mathcal{X}_{\tau(i)}$ be the element of $\mathcal{X}$ that contains $\mathcal{X}_i'$. Finally, let $\mathcal{I}'$ be the largest subset of $\mathcal{X}'$ such that:

$$\forall i \in \mathcal{I}',\; \{j \mid \mathcal{X}_i' \subset \mathcal{W}_j'\} \;=\; \{j \mid \mathcal{X}_{\tau(i)} \subset \mathcal{W}_j\}.$$

If $\mathcal{X}_i'$ does not belong to $\mathcal{I}'$, there exists a chunk $\mathcal{W}_j$ such that some piece of work in $\mathcal{X}_{\tau(i)}$ belongs to $\mathcal{W}_j$ but not to $\mathcal{W}_j'$: $\mathcal{X}_i' \subset \mathcal{W}_j \setminus \mathcal{W}_j'$, $\cup_{i \notin \mathcal{I}'} \mathcal{X}_i' \subset \cup_{i=1}^n \mathcal{W}_i \setminus \mathcal{W}_j'$, and

$$\sum_{i \notin \mathcal{I}'} |\mathcal{X}_i'| = \left| \bigcup_{i \notin \mathcal{I}'} \mathcal{X}_i' \right| \leq \left| \bigcup_{i=1}^n \mathcal{W}_i \setminus \mathcal{W}_j' \right| \leq \sum_{i=1}^n |\mathcal{W}_i \setminus \mathcal{W}_j'| \leq n\varepsilon.$$

Because Σ' implicitly specifies a ($\leq n$)-chunk schedule under the charged-initiation model, its expected work production cannot exceed that of the best such schedule; i.e., $E_n^{(\mathrm{c})}(W_{(\mathrm{ttl})}) \geq E^{(\mathrm{c})}(W_{(\mathrm{ttl})}, \Sigma')$. However, a lengthy calculation that appears in the research report [7] that underlies this paper shows that $E^{(\mathrm{c})}(W_{(\mathrm{ttl})}, \Sigma') \geq E_n^{(\mathrm{f})}(W_{(\mathrm{ttl})}) - n\varepsilon$, whence the righthand bound in Equation (3).

[5] Clearly, having workers attempt all n chunks cannot decrease the overall expectation.

3 Scheduling a Single Worker

We now derive exactly or asymptotically optimal schedules for the single-worker scenario.

3.1 One Worker under the Free-Initiation Model

Under our simpler model, we achieve rather strong results: an exactly optimal schedule under the linear risk function (Section 3.1.1) and an asymptotically optimal schedule for arbitrary risk functions (Section 3.1.2).

3.1.1 An optimal schedule for the linear risk model

Even this simplest scenario, in which there is only one worker and additional chunks of work (which betoken additional communications and/or checkpoints) incur no cost, has complicating subtleties. When we instantiate the expectations derived in Section 2.2.2 with the linear risk function, we discover the following values for the expected amount of completed work when $W_{(\mathrm{ttl})} \leq 1/\kappa$. (1) When one deploys the entire workload as a *single chunk,* $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_1) = W_{(\mathrm{ttl})} - \kappa(W_{(\mathrm{ttl})})^2$. (2) When one deploys the workload as *two chunks* of respective sizes $\omega_1 > 0$ and $\omega_2 > 0$ ($\omega_1 + \omega_2 = W_{(\mathrm{ttl})}$), $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_2) = W_{(\mathrm{ttl})} - W_{(\mathrm{ttl})}^2\kappa + \omega_1\omega_2\kappa$. Thus, $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_2) - E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_1) = \omega_1\omega_2\kappa > 0$, meaning that one increases the expectation of $\mathsf{W}_{(\mathrm{cmp})}$ by deploying any fixed workload as two chunks rather than as one, *no matter how one sizes the chunks.* This trend continues indefinitely: the optimal expectation of $\mathsf{W}_{(\mathrm{cmp})}$ strictly increases with the number of chunks allowed (Theorem 2). This reveals a weakness of the free-initiation model: the (unattainable) "optimal" strategy would deploy infinitely many infinitely small chunks. Stated formally:

Theorem 2 (Optimal schedule: free initiation, linear risk)
The goal. *To deploy $W_{(\mathrm{ttl})}$ units of work to a single worker in at most $n > 0$ chunks, in a way that maximized the expectation of $\mathsf{W}_{(\mathrm{cmp})}$.*
The unique optimal schedule. *Deploy $W_{(\mathrm{dpl})} = \min\left\{W_{(\mathrm{ttl})}, \frac{n}{n+1}X\right\}$ units of work, with each chunk comprising $\frac{1}{n}W_{(\mathrm{dpl})}$ units of work.*
In expectation, this schedule completes $E_n^{(\mathrm{f})}(W_{(\mathrm{ttl})}) = W_{(\mathrm{dpl})} - \frac{1}{2}\left(1 + \frac{1}{n}\right)\kappa W_{(\mathrm{dpl})}^2$ units of work.

Proof We partition the workload into $n+1$ chunks of respective nonnegative sizes $\omega_1, \ldots, \omega_{n+1}$, with the intention of deploying the first n of these.

(a) Assigning the first n chunks *nonnegative,* rather than *positive,* sizes allows us to talk about "at most n chunks" using only the single parameter n.
(b) Creating $n+1$ chunks, rather than n, allows us to hold back work, to avoid the certain interruption of the nth chunk. Exercising this option

means making ω_{n+1} positive; declining the option—thereby deploying all $W_{(\mathrm{ttl})}$ units of work—means setting $\omega_{n+1} = 0$.

Each such partition specifies an n-chunk schedule Σ_n. Our challenge is to choose the sizes of the $n+1$ chunks in a way that maximizes $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_n)$. To simplify notation, let $Z = \omega_1 + \cdots + \omega_n$ denote the portion of the entire workload that we actually deploy. Extending the reasoning from the cases $n = 1, 2$, one obtains easily from (2) the expression

$$\begin{aligned} E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_n) &= \omega_1(1 - \omega_1\kappa) + \cdots + \omega_n(1 - (\omega_1 + \cdots + \omega_n)\kappa) \\ &= Z - Z^2\kappa + \left[\textstyle\sum_{1 \le i < j \le n} \omega_i\omega_j\right]\kappa. \end{aligned} \tag{4}$$

Standard arguments show that the bracketed sum in (4) is maximized when all ω_i's share the common value Z/n, so that the sum never exceeds $\frac{1}{n^2}\binom{n}{2}\kappa Z^2$. Since maximizing the sum also maximizes $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_n)$, we have $E^{(\mathrm{f})}(W_{(\mathrm{ttl})}, \Sigma_n) = Z - \frac{n+1}{2n}\kappa Z^2$.

Because this expression, as a function of Z, is unimodal, increasing until $Z = \frac{n}{(n+1)\kappa}$ and decreasing thereafter, the proof is complete.

Note. Many risk functions, such as the linear risk function, have "built-in" ends to worksharing episodes, because there is an amount of work, call it V, by whose completion, the worker is *certain* to have been interrupted (with probability 1). For such risk functions, one can improve the equal-chunk schedule Σ_n by choosing chunk sizes based on V instead of $W_{(\mathrm{ttl})}$. This phenomenon is visible in Theorem 2 and recurs in our study.

3.1.2 An asymptotically optimal schedule under arbitrary risk

The following notation is useful throughout the paper. Say that our workload consists of $W_{(\mathrm{ttl})}$ units of work that we somehow order linearly. We denote by $\langle a, b\rangle$ the sub-workload obtained by eliminating: the initial a units of work and all work beyond the initial b units. For illustration: $\langle 0, W_{(\mathrm{ttl})}\rangle$ denotes the entire workload; $\langle 0, \frac{1}{2}W_{(\mathrm{ttl})}\rangle$ denotes the first half of the workload; $\langle \frac{1}{2}W_{(\mathrm{ttl})}, W_{(\mathrm{ttl})}\rangle$ denotes the last half of the workload.

We show now that, for one worker, the family of schedules

$$\{\Sigma(n) \mid \Sigma(n) \text{ deploys work in } n \text{ equal-size chunks}\}$$

is *asymptotically optimal* for every risk function, in the sense that the expectation of $\mathsf{W}_{(\mathrm{cmp})}$ under these schedules tends to the expectation of an optimal schedule as n grows without bound.

Theorem 3 (Asymptotically optimal schedule: free initiation, arbitrary risk)
The goal. *To deploy $W_{(\mathrm{ttl})}$ units of work to a single worker in at most $n > 0$ chunks, for some $n > 0$, in a way that maximizes the expectation of $\mathsf{W}_{(\mathrm{cmp})}$.*
An asymptotically optimal schedule, $\Sigma(n)$. *Partition the workload into n equal-size chunks $\mathcal{W}_1, \cdots, \mathcal{W}_n$:*

$$\textit{For each } i \in [1, n], \quad \textit{assign} \quad \left[\mathcal{W}_i \leftarrow \left\langle \frac{i-1}{n}W_{(\mathrm{ttl})}, \frac{i}{n}W_{(\mathrm{ttl})} \right\rangle\right]$$

Proof (Intuition) We compare $\Sigma(n)$'s expected work-production with that of an optimal schedule, $Opt(n)$. We "align" the chunks of $\Sigma(n)$ with those of $Opt(n)$ via two mappings: for each chunk-index i:

1. $s(i)$ is the index of the first chunk of $\Sigma(n)$ that starts no sooner than the end of the ith chunk of $Opt(n)$;
2. $p(i)$ is the index of the last chunk of $\Sigma(n)$ that ends no later than the beginning of the ith chunk of $Opt(n)$.

With this crude alignment, we can bound the deviation of the expected $\Sigma(n)$'s work-production from that of $Opt(n)$, and we find that this deviation becomes negligible as n grows without bound. We refer to [7] for the detailed proof.

3.2 The Charged-Initiation Model, with Linear Risk

The *charged-initiation* analogue of Theorem 2 is drastically more difficult to deal with. The following result is strikingly similar to an analogous result in [10], despite substantive differences between our model and theirs.

Theorem 4 (Optimal schedule: charged initiation, linear risk)
The goal. *To deploy $W_{(\mathrm{ttl})}$ units of work to a single worker in at most $n > 0$ chunks, when $X \geq \varepsilon$, in a way that maximizes $E_n^{(\mathrm{c})}(W_{(\mathrm{ttl})})$.*
The unique optimal schedule. *Let $n_1 = \left\lfloor \left(\sqrt{1 + 8X/\varepsilon} - 1 \right) /2 \right\rfloor$, and let $n_2 = \left\lfloor \left(\sqrt{1 + 8W_{(\mathrm{ttl})}/\varepsilon} + 1 \right) /2 \right\rfloor$.*
1. Deploy the work in $m = \min\{n,\ n_1,\ n_2\}$ chunks;
2. give the first chunk size $\omega_{1,m} = W_{(\mathrm{dpl})}/m + (m-1)\varepsilon/2$, where

$$W_{(\mathrm{dpl})} \ = \ \min\left\{ W_{(\mathrm{ttl})},\ \frac{m}{m+1} X - m\varepsilon/2 \right\}. \tag{5}$$

3. inductively, give the $(i+1)$th chunk size $\omega_{i+1,m} = \omega_{i,m} - \varepsilon$.
In expectation, this schedule completes

$$E_n^{(\mathrm{c})}(W_{(\mathrm{ttl})}) = W_{(\mathrm{dpl})} - \frac{m+1}{2m} W_{(\mathrm{dpl})}^2 \kappa - \frac{m+1}{2} \left(W_{(\mathrm{dpl})} - \frac{(m-1)m}{12} \varepsilon \right) \varepsilon\kappa \tag{6}$$

units of work.

Note that $E_n^{(\mathrm{c})}(W_{(\mathrm{ttl})})$ is maximal for any value of $n \geq \min\{n_1, n_2\}$.

Proof (Intuition) The proof proceeds by induction on the number n of chunks we want to partition our $W_{(\mathrm{ttl})}$ units of work into. This strategy is tractable because our risk model ensures that the only influence the first n chunks of work have on the probability that the last chunk will be computed successfully is in terms of their cumulative size. This means that once one has specified the cumulative size, A_n, of the workload deployed in the first n chunks, the best way to partition this workload into chunks is as though it were the only work in the system, i.e., as if there were no $(n+1)$th chunk to be allocated. The

rather formidable details of the analysis contain several branches based on the sizes of $W_{(\mathrm{ttl})}$ and A_n relative to the expected time of certain interruption X and the per-chunk overhead ε. We refer to [7] for the detailed proof.

About arbitrary risk. Even under the free-initiation model, we were able to achieve only asymptotically optimal schedules for arbitrary risk functions, so we can expect no better under the charged-initiation model.

4 Scheduling Two Workers under the Free Initiation Model

When scheduling a single worker, one need not cope with many of the hardest aspects of scheduling many workers, notably those involving replicating and ordering work. When scheduling *two* workers, though, one does have to cope with such issues, albeit in a simpler setting than the general case. Accordingly, our work in this section does not extend directly to the general case of p workers, but it does allow us to derive principles that guide us as we address the general case in the next section. We focus here only on the free-initiation model, in order to simplify the problem of work replication. By Theorem 1, the optimal schedules that we derive are asymptotically optimal under the charged-initiation model.

We begin with a result that narrows the search for optimal schedules by identifying three characteristics that we may insist on in any candidate optimal schedule for two workers (Section 4.1). We then observe these characteristics "in action" as we derive explicit schedules that are asymptotically optimal under arbitrary risk functions (Section 4.2). We finally refine the latter schedules for the linear risk model, developing schedules that are always at least asymptotically optimal and are exactly optimal under certain circumstances (Section 4.3). A major lesson from this section is that it is much harder to share work optimally with multiple workers than with a single worker.

We only give proof sketches for all results in this section because of the unusual length of the proofs. We refer to [7]. for full details.

4.1 Guidelines for Crafting Optimal Schedules

Say that we are scheduling two workers, P_1 and P_2. For $j = 1, 2$, we deploy a sequence of n_j chunks of work, $\mathcal{W}_{j,1}, \ldots, \mathcal{W}_{j,n_j}$, on worker P_j, insisting that P_j schedules its chunks in the indicated order. We do not assume any *a priori* relation between how P_1 and P_2 break their allocated work into chunks; in fact, work that is allocated to both P_1 and P_2 (what we later call "replicated" work) may be chunked differently on the two computers. Absent more information about the risk function that governs the current worksharing episode, we are not yet ready to prescribe the best way to schedule the work on P_1 and P_2; however, we are able to provide valuable guidelines for this scheduling.

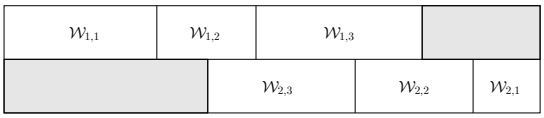


Fig. 1 The shape of an optimal schedule for two workers; cf. Theorem 5 with $n_1 = n_2 = 3$. The top row displays P_1's chunks, the bottom row P_2's. Vertically aligned parts of chunks correspond to replicated work; shaded areas depict unallocated work (e.g., no work from $\mathcal{W}_{2,1}$ is allocated to P_1).

Theorem 5 (Guidelines: nondecreasing risk) *We want to schedule two workers, P_1 and P_2, whose common risk function never decreases as a worker processes more work. Given any schedule Σ for the workers, there exists a schedule Σ' that, in expectation, completes as much work as Σ and that enjoys the following characteristics (see Fig. 1).*

1. Maximal work deployment. *Σ' deploys as much of the workload as possible. Therefore, the workloads that Σ' deploys to P_1 and P_2 can overlap only if their union is the entire workload.*
2. Local-work priority. *Σ' has P_1 (resp., P_2) process all allocated work that it does* not *share with P_2 (resp., with P_1) before processing any shared work.*
3. Replicated work "mirroring". *Σ' has P_1 and P_2 process their shared work "in opposite orders." In detail: Say that P_1 chops its allocated work into chunks $\mathcal{W}_{1,1}, \ldots, \mathcal{W}_{1,n_1}$, while P_2 chops its work into chunks $\mathcal{W}_{2,1}, \ldots, \mathcal{W}_{2,n_2}$. Say that there exist chunk-indices a_1, $b_1 > a_1$ for P_1, and a_2, $b_2 > a_2$ for P_2 such that: chunks $\mathcal{W}_{1,a_1}$ and $\mathcal{W}_{2,a_2}$ both contain a shared "piece of work" A, and chunks $\mathcal{W}_{1,b_1}$ and $\mathcal{W}_{2,b_2}$ both contain a shared "piece" B. Then if Σ' has P_1 execute A before B (i.e., it executes chunk $\mathcal{W}_{1,a_1}$ before chunk $\mathcal{W}_{1,b_1}$, then it has P_2 execute B before A (i.e., it executes chunk $\mathcal{W}_{2,b_2}$ before chunk $\mathcal{W}_{2,a_2}$).*

Proof (Sketch) We devise a cut-and-paste argument for each of the theorem's three characteristics. For each characteristic C in turn: we start with a schedule $\Sigma^{(0)}$ that, in expectation, completes $E^{(0)}$ units of work and that does not have characteristic C. Proceeding inductively, if we currently have a schedule $\Sigma^{(r)}$ that completes $E^{(r)}$ units of work, then we convert $\Sigma^{(r)}$ to obtain a schedule $\Sigma^{(r+1)}$ that, in a sense, comes closer to enjoying characteristic C and that, in expectation, completes $E^{(r+1)} \geq E^{(r)}$ units of work. We prove that a finite sequence of such alterations convert $\Sigma^{(0)}$ to a schedule Σ that enjoys characteristic C. For *maximal work deployment*, we begin with a schedule $\Sigma^{(0)}$ that replicates some work even though it fails to deploy other work. For *local work priority*, we begin with a schedule $\Sigma^{(0)}$ that has a worker process a shared piece of work before processing a local piece. For *shared-work "mirroring"*, we begin with a schedule $\Sigma^{(0)}$ that processes two pieces of shared work in an un-mirrored fashion.

4.2 Scheduling Two Workers under Arbitrary Risk

The results in Section 3 suggest that finding exactly optimal schedules for two workers will be surprisingly difficult, even in simple cases such as single-chunk allocation or total-workload deployment. Moreover, simulations in subsequent sections suggest that simple regular heuristic schedules often complete, in expectation, almost as much work as do complex exactly optimal ones. These facts lead us to abandon a quest for exactly optimal schedules, in favor of asymptotically optimal ones with simple structures. With this more modest goal, we can apply reasoning analogous to that used when analyzing a single worker to adapt the asymptotically optimal schedule $\Sigma(n)$ of Theorem 3—in the light of the guidelines of Theorem 5—to achieve asymptotically optimal expected work-production with two workers.

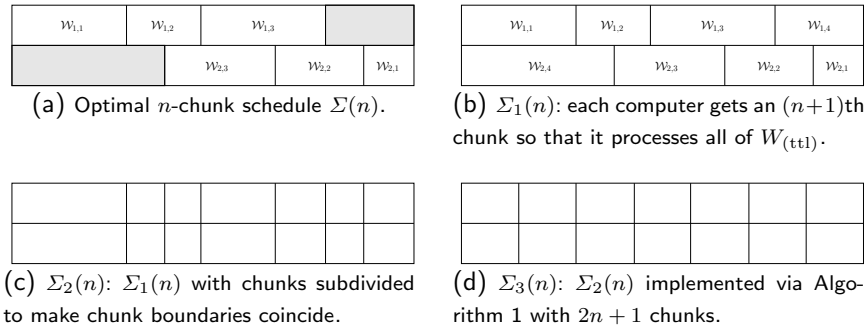
Theorem 6 (Asymptotically optimal schedule: arbitrary risk)
The goal. *To deploy $W_{(\mathrm{ttl})}$ units of work to two workers, in at most $n > 0$ chunks, while maximizing the expectation of $\mathsf{W}_{(\mathrm{cmp})}$.*
An asymptotically optimal schedule. *Allocate the same set of equal-size chunks to both workers, in the following "mirrored" manner:*

$$(\forall i \in [1, n]) \left[\mathcal{W}_{1,i}, \mathcal{W}_{2,n-i+1} \leftarrow \left\langle \frac{i-1}{n} W_{(\mathrm{ttl})}, \frac{i}{n} W_{(\mathrm{ttl})} \right\rangle \right].$$

Proof (Sketch) Denote by $\mathcal{S}_n$ the n-chunk schedule described in the theorem—whose asymptotic optimality we wish to establish. Let Σ_n be an optimal schedule that deploys work in $\leq n$ chunks. Theorem 5 affords us a valid "shape" for Σ_n: We lose no generality by positing that it has the shape depicted in Fig. 2(a) and that it satisfies the three properties of Theorem 5. From this base, we transform Σ_n to schedule $\Sigma_n^{(1)}$—see Fig. 2(b)—by adding a possibly empty $(n+1)$th chunk to each worker's load so that each processes the entire workload; this clearly cannot decrease expected work production. Next, we transform $\Sigma_n^{(1)}$ to $\Sigma_n^{(2)}$ by subdividing chunks so that both workers' chunk boundaries coincide—see Fig. 2(c); we use a simple procedure that is guaranteed not to decrease expected work production. We finally verify that, as n grows without bound, the expected work productions of $\mathcal{S}_n$ (i.e., $E(W_{(\mathrm{ttl})}, \mathcal{S}_n)$) and $\Sigma_n^{(2)}$ (i.e., $E(W_{(\mathrm{ttl})}, \Sigma_n^{(2)})$) tend to the same limit.

4.3 Scheduling Two Workers under Linear Risk

We finally specialize from arbitrary risk functions to linear risk, in order to get stronger performance guarantees under certain conditions. We restrict attention here to schedules for two workers, P_1 and P_2, that are *symmetric*, in the sense that the workers receive the same number of chunks of work, and their local schedules are "translations" of one another. Hence if P_1 subdivides its allocated work into a sequence of pieces $\mathcal{X}_{1,1}, \ldots, \mathcal{X}_{1,m}$ that it processes in that order, then P_2 subdivides its allocated work into a sequence of



(a) Optimal n-chunk schedule $\Sigma(n)$.

(b) $\Sigma_1(n)$: each computer gets an $(n+1)$th chunk so that it processes all of $W_{(\mathrm{ttl})}$.

(c) $\Sigma_2(n)$: $\Sigma_1(n)$ with chunks subdivided to make chunk boundaries coincide.

(d) $\Sigma_3(n)$: $\Sigma_2(n)$ implemented via Algorithm 1 with $2n+1$ chunks.

Fig. 2 Schedule transformations used to prove asymptotic optimality results in Theorems 6 and 8 (Algorithm 1 when $W_{(\mathrm{ttl})} \leq X$).

same-size pieces $\mathcal{X}_{2,1}, \ldots, \mathcal{X}_{2,m}$ that it processes in that order, starting form the other end of the workload. If $\mathcal{X}_{1,1} = \langle 0, w_1 \rangle$ and $\mathcal{X}_{1,2} = \langle w_1, w_2 \rangle$, then $\mathcal{X}_{2,1} = \langle W_{(\mathrm{ttl})} - |w_1|, W_{(\mathrm{ttl})} \rangle$ and $\mathcal{X}_{2,1} = \langle W_{(\mathrm{ttl})} - |w_2|, W_{(\mathrm{ttl})} - |w_1 \rangle$ (and so on). Focusing on symmetric schedules is natural in the absence of extra information that distinguishes among the workers.

4.3.1 Exactly optimal single-chunk schedules

Theorem 7 (Optimal schedule: linear risk, single-chunk deployment)
The goal. *To deploy $W_{(\mathrm{ttl})}$ units of work on two workers, P_1, P_2, deploying a single chunk on each.*
The optimal symmetric schedule Σ.

- *If $W_{(\mathrm{ttl})} \leq \frac{1}{2}X$, then deploy the entire workload on both P_1 and P_2: $\mathcal{W}_{1,1} = \mathcal{W}_{2,1} = \langle 0, W_{(\mathrm{ttl})} \rangle$.*
- *If $\frac{1}{2}X < W_{(\mathrm{ttl})} \leq X$, then deploy the first half of the workload on P_1 and the second half on P_2: $\mathcal{W}_{1,1} = \langle 0, \frac{1}{2}W_{(\mathrm{ttl})} \rangle$, and $\mathcal{W}_{2,1} = \langle \frac{1}{2}W_{(\mathrm{ttl})}, W_{(\mathrm{ttl})} \rangle$.*
- *If $X < W_{(\mathrm{ttl})}$, then deploy only X units of the workload, allocating the first half to P_1 and the second half to P_2: $\mathcal{W}_{1,1} = \langle 0, \frac{1}{2}X \rangle$, and $\mathcal{W}_{2,1} = \langle \frac{1}{2}X, X \rangle$.*

Proof (Sketch) We branch on whether schedule Σ deploys disjoint workloads on the two workers or not. If Σ deploys *disjoint* workloads on P_1 and P_2, then the independence of the workloads allows us to invoke Theorem 2 to discover their optimal sizes. The optimal strategy deploys $W_{(\mathrm{dpl})} = \min\{W_{(\mathrm{ttl})}, X\}$ units of work in total. If Σ deploys *overlapping* workloads on P_1 and P_2, then, to avoid the risk of certain interruption, it never deploys a full X units of work on either worker—so we lose no generality by assuming that $W_{(\mathrm{ttl})} < 2X$. Because we consider only symmetric schedules, Theorem 5 tells us that the common size s of the allocations to P_1 and P_2 satisfies $s \geq W_{(\mathrm{ttl})}/2$. The remainder of the proof branches on the relative values of $W_{(\mathrm{ttl})}$ and X in order to maximize $E(W_{(\mathrm{ttl})}, \Sigma)$.

4.3.2 (Asymptotic) optimal schedules for multi-chunk allocations

This section presents and analyzes Algorithm 1, which prescribes schedules for two workers that are *always asymptotically optimal* and that are *exactly optimal when* $W_{(\text{ttl})} \geq 2X$. We use the notation Algorithm 1(n) to denote the n-chunk invocation of the algorithm. The algorithm employs our notation $\langle a, b \rangle$ for sub-workloads from the beginning of Section 3.1.2.

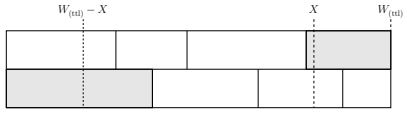
Algorithm 1: Scheduling for 2 workers deploying at most n chunks per worker. The value n is an input.

if $W_{(\text{ttl})} \geq 2X$ **then**
  $\forall i \in [1,n]\ \mathcal{W}_{1,i} \leftarrow \left\langle \frac{i-1}{n} \times \frac{n}{n+1}X,\ \frac{i}{n} \times \frac{n}{n+1}X \right\rangle$
  $\forall i \in [1,n]\ \mathcal{W}_{2,i} \leftarrow \left\langle W_{(\text{ttl})} - \frac{i}{n} \times \frac{n}{n+1}X,\ W_{(\text{ttl})} - \frac{i-1}{n} \times \frac{n}{n+1}X \right\rangle$
if $W_{(\text{ttl})} \leq X$ **then**
  $\forall i \in [1,n]\ \mathcal{W}_{1,i} = \mathcal{W}_{2,n-i+1} \leftarrow \left\langle \frac{i-1}{n}W_{(\text{ttl})},\ \frac{i}{n}W_{(\text{ttl})} \right\rangle$
if $X < W_{(\text{ttl})} < 2X$ **then**
  $\ell \leftarrow \lfloor n/3 \rfloor$
  $\forall i \in [1,\ell]\ \mathcal{W}_{1,i} \leftarrow \left\langle \frac{i-1}{\ell}(W_{(\text{ttl})} - X),\ \frac{i}{\ell}(W_{(\text{ttl})} - X) \right\rangle$
  $\forall i \in [1,\ell]\ \mathcal{W}_{2,i} \leftarrow \left\langle W_{(\text{ttl})} - \frac{i}{\ell}(W_{(\text{ttl})} - X),\ W_{(\text{ttl})} - \frac{i-1}{\ell}(W_{(\text{ttl})} - X) \right\rangle$
  $\forall i \in [1,2\ell]\ \mathcal{W}_{1,\ell+i} = \mathcal{W}_{2,3\ell-i+1} \leftarrow$
  $\left\langle (W_{(\text{ttl})} - X) + \frac{i-1}{2\ell}(2X - W_{(\text{ttl})}),\ (W_{(\text{ttl})} - X) + \frac{i}{2\ell}(2X - W_{(\text{ttl})}) \right\rangle$

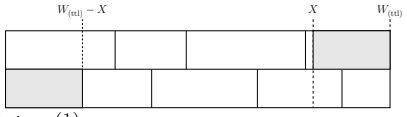
Theorem 8 ((Asymptotically) optimal schedule: linear risk)
The schedules specified by Algorithm 1 have the following performance.

1. *When* $W_{(\text{ttl})} \geq 2X$, *they are* exactly optimal, *completing, in expectation,* $(1 - \frac{1}{n})X$ *units of work.*
2. *When* $W_{(\text{ttl})} < 2X$, *they are* asymptotically optimal.
   (a) *When* $W_{(\text{ttl})} \leq X$, *their expected work production tends asymptotically to* $W_{(\text{ttl})} - \frac{1}{6}W_{(\text{ttl})}^3\kappa^2$ *units.*
   (b) *When* $X < W_{(\text{ttl})} < 2X$, *their expected work production tends asymptotically to* $2W_{(\text{ttl})} - \frac{1}{3}X - W_{(\text{ttl})}^2\kappa + \frac{1}{6}W_{(\text{ttl})}^3\kappa^2$ *units.*

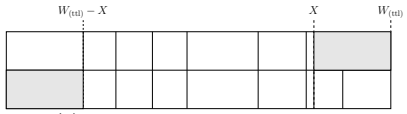
Proof (Sketch) When $W_{(\text{ttl})} \geq 2X$, Theorem 5 tells us that we lose no work production by insisting that the workers work on disjoint subsets of the workload; Theorem 2 tells us how to schedule the subworkloads, and it gives the expectation of $\mathsf{W}_{(\text{cmp})}$. When $W_{(\text{ttl})} < 2X$, we focus on a fixed but arbitrary number n of chunks and on an optimal n-chunk schedule Σ_n that (with no loss of generality) enjoys the three characteristics of Theorem 5. Depending on the exact relationship between $W_{(\text{ttl})}$ and X, we consider two sequences



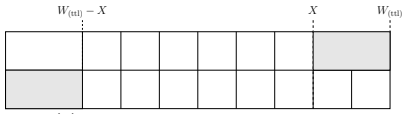
(a) Σ_n: Optimal schedule with n chunks.



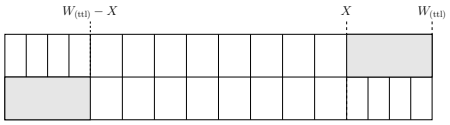
(b) $\Sigma_n^{(1)}$: Adding an $(n+1)$th chunk to extend Σ_n's subworkloads to size X.



(c) $\Sigma_n^{(2)}$: Subdividing chunks to line up boundaries.



(d) $\Sigma_n^{(3)}$: Equalizing chunk sizes in the intervals.



(e) $\Sigma_n^{(4)}$: Algorithm 1's schedule with $3(n+1)$ chunks.

Fig. 3 Schedule transformations that prove the asymptotic optimality of Algorithm 1 when $X < W_{(\mathrm{ttl})} < 2X$.

of transformations that gradually convert Σ_n into a schedule Σ that Algorithm 1 produces—but that deploys more than n chunks. Specifically: For the case $W_{(\mathrm{ttl})} < X$, the transformations of Fig. 2 produce a Σ that deploys $2n+1$ chunks. For the case $X < W_{(\mathrm{ttl})} < 2X$, the transformations of Fig. 3 produce a Σ that deploys $3(n+1)$ chunks. In both cases, we are able to show that $E(W_{(\mathrm{ttl})}, \Sigma_n)$ and $E(W_{(\mathrm{ttl})}, \Sigma)$ tend to the same limit as n grows without bound.

5 Strategies for Crafting General Schedules

This section is devoted to developing strategies for crafting simple heuristic schedules for sharing work with arbitrary numbers of workers. In an attempt to describe our heuristics in terms that are clear, evocative, and free of undesired connotations, we introduce the following terminology.

- A *coterie* is a collection of workers that work jointly on the same subset of our workload.
- A *slice* of our workload is a subset that we assign to a single coterie.

All of our scheduling heuristics operate as follows: (i) they *partition* the total workload into slices that they deploy on disjoint coteries; (ii) they tell each coterie how to partition its assigned slices into *equal-size* chunks; and (iii) they orchestrate the processing of the slices on each coterie.

The entire scheduling process is performed in the light of our interruption model. Specifically, we acknowledge the futility of deploying a slice of work of

size $> X$ on any worker, because such a slice is "certain" to be interrupted. The amount of work that we actually deploy to the workers, which we denote by $W_{(\mathrm{dpl})}$, is, therefore, often less than the $W_{(\mathrm{ttl})}$ units of work in the total workload. Part of the scheduling challenge is to determine the size of $W_{(\mathrm{dpl})}$.

5.1 The Partitioning Phase

We begin to develop our schedules by specifying simple heuristics for determining a value for $W_{(\mathrm{dpl})}$ and for partitioning this amount of work into slices for deployment on the workers. While our heuristics are tailored for the linear risk function, we show how to adapt them to other risk functions. Our scheduling strategy branches into substrategies based on the size of the workload.

A. $W_{(\mathrm{ttl})}$ is "very large." When $W_{(\mathrm{ttl})} \geq pX$, we deploy p slices, each of size X, to be processed independently by all workers. We abandon the remaining $W - pX$ units of work, because our interruption model tells us that it is "certain" not to be completed. (Work is not prioritized, so we do not care *which* pX units get done.) We schedule the deployed work on each worker in the manner prescribed by the single-worker guidelines of Section 3.

B. $W_{(\mathrm{ttl})}$ is "very small." When $W_{(\mathrm{ttl})} \leq X$, we allocate the entire workload as a single slice, which we replicate on all p workers. We thus have a single coterie comprising all p workers, working on a slice of size X; we orchestrate the work in the manner explained in Section 5.2.

C. $W_{(\mathrm{ttl})}$ is of "intermediate" size. The case $X < W_{(\mathrm{ttl})} < pX$ is the interesting challenge, as there is no compelling known scheduling strategy. Here we can deploy the total $W_{(\mathrm{ttl})}$-unit workload, replicating each deployed piece $pX/W_{(\mathrm{ttl})}$ times on average. (Note that no worker receives more than X units of work with this scheme.) In our quest for schedules that are simple to implement, we assign disjoint coteries to work on independent slices of work. Since the number of coteries must be integral, we approximate the ideal of $W_{(\mathrm{dpl})}\kappa$ coteries by partitioning the workload into $q = \lceil W_{(\mathrm{dpl})}\kappa \rceil$ slices. We balance computing resources as much as possible, by replicating each slice on either $\lfloor p/q \rfloor$ or $\lceil p/q \rceil$ workers. We balance the load among the coteries by having a coterie with $\lfloor p/q \rfloor$ workers (resp., $\lceil p/q \rceil$ workers) work on a slice of size $sl^- = \lfloor p/q \rfloor \, W_{(\mathrm{dpl})}/p$ (resp., of size $sl^+ = \lceil p/q \rceil \, W_{(\mathrm{dpl})}/p$).

Accommodating general risk functions. The preceding scenario is tailored for the linear risk function in that the load of each worker has size $\leq X$. To adapt the strategy to general risk functions, we introduce a parameter λ that specifies the maximum probability of interruption that the user would allow for the work allocated to a worker. (For linear risk, $\lambda = 1$, a choice that could often be impractical; consider, e.g., a heavy-tailed distribution.) We use λ to compute the maximum size of a slice, $maxsl$, by insisting that $Pr(maxsl) = \lambda$. For illustration, if $\lambda = 1/2$, then under the linear risk function we would set $maxsl = \frac{1}{2}X$, while under the exponential risk function we would set $maxsl = (\ln 2)X$. The deployed workload would consist of $W_{(\mathrm{dpl})} = \min(W_{(\mathrm{ttl})},\ p \times$

$maxsl$) units, which mandates using $q = \lceil W_{(\mathrm{dpl})}/maxsl \rceil$ slices whose sizes are as defined earlier.

5.2 The Orchestration Phase

Our partitioning phase has formed mutually disjoint slices of work, each of size sl, that we deploy to disjoint coteries. We partition each slice into equal-size chunks. (Theorems 3 and 6 show, respectively, that equal-size-chunk schedules are asymptotically optimal for one or two workers.) The first issue in orchestrating the deployed work is to resolve what the *checkpointing granularity* should be, as measured by the number of chunks we partition each slice into. We denote this quantity by n, so that each worker partitions its slice into n chunks of common size $\omega = sl/n$. For the well-structured heuristics that we design here, we can actually determine the best value of n. (We must defer this determination until Section 5.2.3.) Each chunk of work that is deployed to a coterie Γ will be scheduled on every one of Γ's g workers. Our challenge is to determine the time-step at which each chunk i will be scheduled on each worker P_j of Γ as we strive for a schedule that maximizes the aggregate expected amount of completed work. We discuss the orchestration phase in detail only for the linear risk model, then generalize this discussion to arbitrary risk functions in Section 7.1.

5.2.1 Overview

We illustrate our approach to orchestration via an example wherein each coterie contains $g = 4$ workers and each slice is partitioned into $n = 12$ chunks. Because coteries operate mutually independently, we need develop a schedule for just one coterie and replicate it on all others. Given a coterie Γ and its associated slice, we represent a possible schedule via an *execution chart C for Γ*, as depicted in Table 1. Each row of chart C represents a worker in Γ; each column represents one of the chunks into which Γ's slice is chopped; chart entry $C_{i,j}$ is the step at which chunk j is processed by worker P_i.

Any $g \times n$ integer matrix whose rows are permutations of $[1..n]$ is a valid execution chart, under which each P_i executes each chunk j precisely once, at step $C_{i,j}$. One can calculate the expected amount of work completed under a chart-specified schedule Σ. use an execution chart to calculate the expected amount of work completed under the schedule Σ that the chart specifies. To

Computer \ Chunk	1	2	3	4	5	6	7	8	9	10	11	12
P_1	1	6	9	12	2	5	8	11	3	4	7	10
P_2	12	1	6	9	11	2	5	8	10	3	4	7
P_3	9	12	1	6	8	11	2	5	7	10	3	4
P_4	6	9	12	1	5	8	11	2	4	7	10	3

Table 1 Execution chart for a general schedule.

wit, chunk j will *not* be executed under Σ only if all g workers in the coterie are interrupted before they complete the chunk. This occurs with probability $\prod_{i=1}^{g} Pr(C_{i,j}\omega) = \prod_{i=1}^{g} Pr\left(C_{i,j}sl/n\right)$, so the expected amount of work completed from the slice is

$$E(sl, n, \Sigma) = \left(1 - \frac{1}{n}\left(\frac{sl \times \kappa}{n}\right)^{g} \sum_{j=1}^{n} \prod_{i=1}^{g} C_{i,j}\right) sl. \tag{7}$$

This last expression is specific to the linear risk model and assumes that

$$Pr(C_{i,j}\omega) = C_{i,j}\omega\kappa \leq 1. \tag{8}$$

We retain assumption (8) throughout this section.

Note. We retain assumption (8) for pragmatic reasons that are justified by our simulations. Under our partitioning strategy, the assumption is true when a coterie has $\lfloor p/q \rfloor$ workers, because $sl^{-} \leq X$, which ignores cases when $sl^{+} > X$. Taking the latter cases into account would considerably complicate all expectation formulas, thereby preventing us from drawing conclusions when comparing heuristics. It is, therefore, convenient to retain the assumption even when we know it does not always hold. Fortunately, the conclusions that we reach in Section 5.3 using this assumption are supported by our experiments, which consider *all* cases (see Section 6 and [7]). The experiments thus provide an *a posteriori* justification for the simplifying assumption. In other words, the performance of the coteries with $\lfloor p/q \rfloor$ workers empirically gives us good insight into the actual performance of heuristics.

Proposition 1 *For any schedule Σ,*

$$E(sl, n, \Sigma) \leq \left(1 - \left(sl \times \kappa \frac{(n!)^{1/n}}{n}\right)^{g}\right) sl \approx \left(1 - \left(\frac{sl \times \kappa}{e}\right)^{g}\right) sl.$$

Proof By Eq. (7), $E(sl, n, \Sigma)$ is maximized when the sum of the n column-products is minimized. Now, the product of the column-products is constant, because each row is a permutation of $[1..n]$. Therefore, the sum of the column-products is minimized when all products are equal (with common value $(n!)^{g/n}$). The claimed inequality follows, with its Stirling approximation.

5.2.2 Group Schedules: Introduction

Under the schedule of Table 1, chunks 1–4 are always executed at the same steps—but by different workers; the same is true for chunks 5–8 as a group and for chunks 9–12 as a group. The twelve chunks of the slice thus partition naturally into three *groups*. One can, therefore, re-specify the schedule of Table 1 as the *group-(oriented)* schedule of Table 2, thereby significantly simplifying the specification. In the group execution chart of Table 2, each column corresponds to a group of chunks, and the ith row specifies the step at which chunks

are executed for the ith time; e.g., each chunk in group 1 is executed for the first time at step 1, for the second time at step 6, and so on. This specification leaves implicit that chunk indices within each group are permuted cyclically at each step, so that each chunk is scheduled on each worker exactly once.

Group 1 chunks 1–4	Group 2 chunks 5–8	Group 3 chunks 9–12
1	2	3
6	5	4
9	8	7
12	11	10

Table 2 Execution chart for a group schedule.

We generalize this description. When n is a multiple of g, we can sometimes convert the full $g \times n$ execution chart C exemplified by Table 1 to the $g \times n/g$ group execution chart $\widehat{C}$ exemplified by Table 2. There are n/g groups of size g, and chart-entry $\widehat{C}_{i,j}$ denotes the step at which group j of chunks is executed for the ith time. For a chart $\widehat{C}$ to specify a valid group schedule Σ, its total set of entries must be a permutation of $[1..n]$. In this case, we denote the chart by $\widehat{C}^{(\Sigma)}$. We can compute easily the expected amount of work that schedule Σ completes under the linear risk model:

$$E(sl, n, \Sigma) \;=\; sl - \mathsf{K}^{(\Sigma)} \times \frac{g}{\kappa} \left(\frac{sl \times \kappa}{n} \right)^{g+1};$$

where $\mathsf{K}^{(\Sigma)} = \sum_{j=1}^{n/g} \prod_{i=1}^{g} \widehat{C}_{i,j}^{(\Sigma)}$ is Σ's *performance constant*. Note that *a smaller value of* $\mathsf{K}^{(\Sigma)}$ *corresponds to a larger value of* $E(sl, n, \Sigma)$.

Group schedules are very natural, because they are *symmetric:* all workers are allocated the same number of chunks, and each worker's schedule is a "translate" of each other's; i.e., each worker executes a chunk from the ith group at the same steps as every other worker. Intuition suggests that the most productive schedules are symmetric: Why should some of the identical workers be treated differently from others by a "nature" that interrupts all workers at random times (within the distribution specified by the risk function)? Lending perspective to Eq. (7) and confidence in the upper bound of Prop. 1—and, more generally, in our focus on group schedules—is the fact that we have not been able to strengthen the latter bound for group schedules! In fact, the latter bound affords us an easy lower bound on the performance constant of any group schedule that has parameters g and n: $\mathsf{K}_{\min} = \lceil (n/g)(n!)^{g/n} \rceil$. We shall see in Section 5.2.4 that this value of $\mathsf{K}_{\min}$ cannot be improved in general.

5.2.3 Choosing the granularity of checkpointing

Happily, at least for group schedules, one often does not have to guess at the best number n of chunks to partition each worker's slice into. We begin to flesh

out this remark by citing from [7] an explicit expression for the expected work completed by any group schedule Σ under any nondecreasing risk function, within the charged-initiation model. One can obtain this expression from Σ's analogous expectation under the free-initiation model.

Theorem 9 *Let Σ be a group schedule defined by the execution chart $C_{i,j}\,|_{i\in\{1,\ldots,g\},j\in\{1,\ldots,n/g\}}$. For any nondecreasing risk function,*

$$E^{(\mathrm{c},n)}(sl^{(\mathrm{c})},\Sigma) \;=\; \frac{sl^{(\mathrm{c})}}{sl^{(\mathrm{c})}+n\varepsilon}E^{(\mathrm{f},n)}(sl^{(\mathrm{c})}+n\varepsilon,\Sigma). \tag{9}$$

We can often use Eq. (9) to determine the optimal value for n, specifically for any risk function under which $E^{(\mathrm{c},n)}(sl^{(\mathrm{c})},\Sigma)$ is a *unimodal* function of n. (It is quite natural to assume that $E^{(\mathrm{f},n)}(sl^{(\mathrm{f})},\Sigma)$, the analogous expectation for the free-initiation model, is *nondecreasing* with n, but with the charged-initiation model, the per-chunk overhead damps the work production.) In such cases, binary search (on the number of chunks per slice) will yield the optimum value of n; the search can safely be performed within the interval $[1..X/\varepsilon]$.

5.2.4 Group Schedules: A Sampler

Our group schedules strive to maximize expected work completion by minimizing the impact of work-killing interruptions; they strive for the latter goal by having every worker attempt to compute every chunk. Of course, there are many ways to achieve this coverage, and the form of the risk function may make some ways more advantageous than others. (Theorem 5 indicates that there can exist "risk-independent" rules, at least for special cases.) We now

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

(a) Cyclic
$\mathsf{K}^{(\Sigma_{\mathsf{cyclic}})} = 34104$

1	2	3	4	5
10	9	8	7	6
15	14	13	12	11
20	19	18	17	16

(b) Reverse
$\mathsf{K}^{(\Sigma_{\mathsf{reverse}})} = 24396$

1	2	3	4	5
6	7	8	9	10
15	14	13	12	11
20	19	18	17	16

(c) Mirror
$\mathsf{K}^{(\Sigma_{\mathsf{mirror}})} = 27284$

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16

(d) Snake
$\mathsf{K}^{(\Sigma_{\mathsf{snake}})} = 25784$

1	2	3	4	5
14	12	10	8	6
15	13	11	9	7
16	17	18	19	20

(e) Fat-snake
$\mathsf{K}^{(\Sigma_{\mathsf{fat\text{-}snake}})} = 24276$

1	2	3	4	5
10	9	8	7	6
15	14	13	12	11
20	19	18	16	17

(f) Greedy
$\mathsf{K}^{(\Sigma_{\mathsf{greedy}})} = 24390$

1	2	3	4	5
13	10	6	9	7
18	15	14	11	8
20	16	19	12	17

(g) Optimal
$\mathsf{K}^{(\Sigma_{\mathrm{Optimal}})} = 23780$

Table 3 Comparing group schedules for $n = 20$ and $g = 4$. Here the most efficient group schedules are $\Sigma_{\mathsf{fat\text{-}snake}}$, Σ_{greedy}, and $\Sigma_{\mathsf{reverse}}$ (in this order). The lower bound, $\mathsf{K}_{\min} = 23780$, is reached on this example.

specify and compare the performance of five group schedules whose chunk-scheduling schedules seem to be a good match for the way the linear risk function "predicts" interruptions. We specify each schedule Σ via its group execution chart $\widehat{C}^{(\Sigma)}$—see Table 3—and we represent Σ's performance via its performance constant $\mathsf{K}^{(\Sigma)}$. The beneficent structures of these schedules allows us to present explicit expressions for their K constants.

Cyclic scheduling (Table 3(a)). Under this simplest regimen, whose schedules we denote Σ_{cyclic}, groups are executed sequentially in a round-robin fashion, so the chunks of group j are executed at steps j, $j+n/g$, $j+2n/g$, and so on. One verifies that

$$\mathsf{K}^{(\Sigma_{\mathsf{cyclic}})} \;=\; \sum_{j=1}^{n/g} \prod_{k=0}^{g-1} (j + kn/g)\,.$$

The weakness of Σ_{cyclic} is that chunks in low-index groups have a higher probability of being completed successfully than do chunks in high-index groups—because chunks remain in the same relative order throughout the computation. The other schedules we consider aim to compensate for this imbalance.

Reverse scheduling (Table 3(b)). Under this regimen, whose schedules we denote $\Sigma_{\mathsf{reverse}}$, each group's chunks are executed once in the initially specified order and then in the *reverse* order $n/g-1$ times. Thus, the chunks in group j are executed at step j and thereafter at steps $2n/g-j+1$, $3n/g-j+1$, $4n/g-j+1$, etc. This regimen strives to compensate for the imbalance in chunks' likelihoods of being completed engendered by their initial order of processing. ($\Sigma_{\mathsf{reverse}}$ is the schedule specified in Table 2.) One verifies that

$$\mathsf{K}^{(\Sigma_{\mathsf{reverse}})} \;=\; \sum_{j=1}^{n/g} j \times \prod_{k=1}^{g-1} ((k+1)n/g - j + 1)\,.$$

Mirror scheduling (Table 3(c)). This regimen, whose schedules we denote Σ_{mirror}, represents a compromise between the cyclic and reverse schedules. Σ_{mirror} compensates for the imbalance in the likelihood of work's being completed only during the second half of the computation. Specifically, Σ_{mirror} mimics Σ_{cyclic} for the first $g/2$ phases of processing a group, and it mimics $\Sigma_{\mathsf{reverse}}$ for the remaining phases. One verifies that

$$\mathsf{K}^{(\Sigma_{\mathsf{mirror}})} \;=\; \sum_{j=1}^{n/g} \prod_{k=0}^{\frac{1}{2}g-1} (j + kn/g)\,((p-k)n/g - j + 1)\,.$$

Snake-like scheduling (Table 3(d)). This regimen, whose schedules we denote Σ_{snake}, compensates for the imbalance of cyclic scheduling by mimicking Σ_{cyclic} at every odd-numbered step and $\Sigma_{\mathsf{reverse}}$ at every even-numbered step, thereby lending a snake-like structure to its execution chart, $\widehat{C}^{(\Sigma_{\mathsf{snake}})}$. One verifies that

$$\mathsf{K}^{(\Sigma_{\mathsf{snake}})} \;=\; \sum_{j=1}^{n/g} \prod_{k=0}^{\frac{1}{2}g-1} (j + 2kn/g)\,(2(k+1)n/g - j + 1)\,.$$

Fat-snake-like scheduling (Table 3(e)). This regimen, whose schedules we denote $\Sigma_{\mathsf{fat\text{-}snake}}$, adopts the same qualitative strategy as Σ_{snake}, but it slows down the latter's reverse phase. Consider, for illustration, three consecutive rows of $\widehat{C}^{(\Sigma_{\mathsf{fat\text{-}snake}})}$. The first is identical in shape to the first row of $\widehat{C}^{(\Sigma_{\mathsf{cyclic}})}$. The reverse phase of Fat-snake distributes the elements of the two remaining rows in the reverse order, two elements at a time. The motivating intuition is that the slower reversal would further compensate for the imbalance in Σ_{cyclic}. One verifies that

$$\mathsf{K}^{(\Sigma_{\mathsf{fat\text{-}snake}})} \;=\; \sum_{j=0}^{n/g-1} \prod_{k=0}^{\frac{1}{3}g-1} (1+j+3k\frac{n}{g})(3(k+1)\frac{n}{g}-2j-1)(3(k+1)\frac{n}{g}-2j).$$

We derive performance bounds for these five scheduling regimens via straightforward bounds based on the preceding K-constants. Details appear in [7].

Proposition 2 *The values of $\mathsf{K}^{(\Sigma)}$ for our five scheduling regimens — presented via the ratio $\mathsf{K}^{(\Sigma)} \div (g!\,(n/g)^{g+1})$ — satisfy the following bounds:*

Schedule	*Lower bound*	*Upper bound*
$\mathsf{K}^{(\Sigma_{\mathsf{cyclic}})}$	$1/n$	1
$\mathsf{K}^{(\Sigma_{\mathsf{reverse}})}$	$1/(2g)$	$\frac{1}{2}(n+g)$
$\mathsf{K}^{(\Sigma_{\mathsf{mirror}})}$	$1/n$	1
$\mathsf{K}^{(\Sigma_{\mathsf{snake}})}$	g/n^2	1
$\mathsf{K}^{(\Sigma_{\mathsf{fat\text{-}snake}})}$	$1/((g-1)n)$	g

Prop. 2 suggests that Σ_{snake} may be the most efficient of our five group-scheduling regimens, especially when we checkpoint often, i.e., when n is large. We note finally that Stirling's formula yields bounds on $\mathsf{K}^{(\Sigma_{\mathsf{snake}})}$ that are more evocative than the bounds in Prop. 2: $\mathsf{K}_{\min} \;\leq\; \mathsf{K}^{(\Sigma_{\mathsf{snake}})} \;\leq\; \mathsf{K}_{\mathsf{upper}}$, where

$$\left[\mathsf{K}_{\min} \;\approx\; \frac{e}{g}\left(\frac{n}{g}\right)^{g+1}\right] \quad\text{and}\quad \left[\mathsf{K}_{\mathsf{upper}} \;=\; g!\left(\frac{n}{g}\right)^{g+1} \;\approx\; \frac{e\sqrt{2\pi}}{\sqrt{g}}\left(\frac{n}{e}\right)^{g+1}\right].$$

We conclude this subsection by adding a last group schedule to our list, as a reference point for our experiments.

Greedy scheduling (Table 3(f)). This regimen, whose schedules we denote Σ_{greedy}, strives to *iteratively* balance the probability of successful completion for each group of chunks. At each step, Σ_{greedy} constructs one new row of its execution chart $\widehat{C}^{(\mathsf{greedy})}$. Thus, after k steps, the probability that a chunk in group j will be interrupted is proportional to the product $\prod_{i=1}^{k} \widehat{C}_{ij}^{(\mathsf{greedy})}$ of the entries in column j of the chart. The idea is to sort current column-products and assign the smallest time-step to the largest product, and so on. We have not been able to derive an analytical estimate of Σ_{greedy}'s (asymptotic) expected work production, so we content ourselves with numerical estimates.

	Relative				Absolute				Success
	min	max	avg.	stdv.	min	max	avg.	stdv.	rate
Cyclic	1.1	3.786	2.143	0.664	1.1	3.786	2.239	0.592	00.00%
Reverse	1	1.295	1.055	0.065	1	1.295	1.117	0.061	12.42%
Mirror	1	2.468	1.504	0.393	1	2.468	1.575	0.338	12.37%
Snake	1	1.199	1.127	0.059	1	1.291	1.193	0.059	12.34%
Fat-snake	1	1.442	1.123	0.115	1	1.530	1.192	0.143	17.07%
Greedy	1	1.055	1.005	0.015	1	1.224	1.067	0.074	83.01%
Best-of	1	1	1	0	1	1.224	1.061	0.069	100.00%

Table 4 Statistics on the K value of all heuristics for $2 \leq g \leq 100$ and $2g \leq n \leq 1000$ (minimum, maximum, average value and standard deviation over the 4032 instances). The success rate of a given heuristic is the percentage of instances where it is the best one.

Table 3 provides a scenario in which Σ_{greedy} outperforms most of the other heuristics. None of our group schedules completes an optimal amount of work in this scenario, which shows that one pays a price for the heuristics' simple structures.

5.3 Evaluating the Heuristics Numerically

We ran all six of our scheduling heuristics on problems having all parameter values in the following ranges: $g \in [2, 100]$, $n \in [2g, 1000]$, and g divides n, altogether 4032 instances. Table 4 summarizes the results of this evaluation via two series of statistics. In the *Relative* series, we form the ratio of the performance constant K of a given heuristic on a given problem instance and (i.e., divided by) the lowest value of K that we found for that instance, among all of the tested heuristics. For the *Absolute* series, we form the analogous ratio with $\mathsf{K}_{\min}$ in the denominator. For perspective, the table also reports the "performance" of the (unrealistic) *best-of* heuristic that, on each problem instance, runs the six other algorithms and picks the best answer. We see from Table 4 that Σ_{greedy} is clearly the best heuristic: it finds the best schedule for 83% of the problem instances, and its schedules are never more than 6% worse than the best one found. More importantly, Σ_{greedy}'s performance is never more than 23% larger than the lower bound $\mathsf{K}_{\min}$ and, on average, it is less than 7% larger than this bound. In fact, only $\Sigma_{\mathsf{fat\text{-}snake}}$ happens sometimes to find better solutions than Σ_{greedy}—but only by marginal amounts, as one can see by comparing the absolute performance of Σ_{greedy} and *best-of*.

6 Experiments Based on the Linear Risk Model

We have assessed the performance of our group scheduling heuristics by testing them on simulated computing platforms that are subject to unrecoverable interruptions. Since the relative performance of the heuristics were consistent with the theoretical predictions (see Table 4), we carry forward only the champion heuristic, Σ_{greedy}, to the discussion that follows. Whereas the heuristics of

Section 5 (including Σ_{greedy}) were tailored for the linear risk model, the heuristics that henceforth compete with Σ_{greedy} are suggested less formally by the way that they implement the strategy of work replication. The source codes and raw data for all heuristics can be found at `http://graal.ens-lyon.fr/~fvivien/Publications/Data/Interruptions`. The research report [7] that underlies this paper presents plots for all the experiments.

6.1 The Experimental Plan

We performed our experiments on randomly generated platforms containing p computers. In all experiments, we set $\kappa = 1$ and chose interruption times uniformly randomly between 0 and 1. We varied the size of the workload, $W_{(\mathrm{ttl})}$, between 1 and p. The case $W_{(\mathrm{ttl})} = 1$ represents the scenario in which all computers can potentially do all the work before being interrupted. The case $W_{(\mathrm{ttl})} = p$ represents the scenario in which we can do no better than give one unit-size slice to each computer, which then computes until it is interrupted; there is no replication in this case.

The key parameters in our experiments are: the number of computers p; the total amount of work $W_{(\mathrm{ttl})}$; the number of chunks per unit of work n; and the start-up cost ε. In the first four series of experiments, three of these parameters are fixed while the fourth is varied. When fixed, these parameters take the following values: $p \in \{5, 10, 25, 50, 100\}$; $W_{(\mathrm{ttl})} = 0.3p$ or $0.7p$; $n \in \{47, 97, 147, 197\}$; and $\varepsilon \in \{0.1000, 0.0100, 0.0010, 0.0001\}$. We have the parameters range over large sets of values in order to assess the heuristics in very different configurations, even very unfavorable ones.

Our experiments compared the performance of the following heuristics:

The group-greedy heuristic Σ_{greedy}. We have already seen Σ_{greedy} in Section 5.2.2. Since the number of chunks n may not be a multiple of g, the last group of computers may not have a full set of g chunks to process. This heuristic works *as though* the last group contained g chunks; this has the potential impact of inserting idle time-slots in a schedule. (A computer that still has work will never be kept idle.) The values for n were explicitly picked *not* to favor the group-scheduling heuristics: they almost certainly ensured that the last group of computers never had a full set of g chunks to process.

The brute-force replication heuristic Σ_{brute}. This heuristic replicates the entire workload onto all computers. Each computer executes work in the order of receipt, starting from the first chunk, until it is interrupted.

The no-replication heuristic $\Sigma_{\mathsf{no\text{-}rep}}$ This heuristic distributes work in a round-robin fashion, with no replication. Thus, each computer is allocated $W_{(\mathrm{ttl})}/p$ units of work.

The cyclic-replication heuristic $\Sigma_{\mathsf{cyclic\text{-}rep}}$ This heuristic distributes the work in a round-robin fashion, as does $\Sigma_{\mathsf{no\text{-}rep}}$, but it keeps distributing chunks, starting from chunk 1 again, until each computer has a total (local) workload of 1. Note that when the number of chunks is a multiple of p, this heuristic

is identical to $\Sigma_{\text{no-rep}}$, because the chunks assigned to a computer during the replication phase were already assigned to it previously.

The random-replication heuristic $\Sigma_{\text{random-rep}}$ This heuristic distributes a total workload of 1 to each computer, but it chooses the chunks and their order randomly, while ensuring that all chunks deployed on each computer are distinct. Note, however, that the same chunk can be assigned to several computers.

The omniscient heuristic $\Sigma_{\text{omniscient}}$ This heuristic is an idealized, unrealizable "heuristic" that is included only as a reference point. The "heuristic" knows (in advance) exactly when each computer will be interrupted. It deploys on each computer a single chunk of length $\ell + \varepsilon$ (recall that ε is the start-up cost); ℓ is chosen so that the computer completes its work immediately before it is interrupted. Thus, this heuristic, knowing all interruption times, completes the maximum possible amount of work.

We do not report the absolute amount of work completed by the heuristics: This quantity would be impossible to interpret because the amount of work deployed and the amount of work that can be completed before computers are interrupted vary vastly within the experiment. We therefore consider, for each instance, the *ratio of the work completed by a given heuristic and the work completed by* $\Sigma_{\text{omniscient}}$. Since $\Sigma_{\text{omniscient}}$ always achieves a performance ratio of 1, we do not display its performance on plots. (In the marginal cases where $\Sigma_{\text{omniscient}}$ does not complete any work—because all computers are interrupted by time ε—the performance of all heuristics is set to 1.)

6.2 Results from Idealized Experiments

For each considered set of parameters, we randomly generated 1000 different sets of interruption times. In Experiments E1–E4, we fixed all parameters but one; we varied, respectively: (E1) the workload size $W_{(\text{ttl})}$, (E2) the number of computers p, (E3) the number of chunks per unit size n, (E4) the start-up cost ε. Fig. 4 presents the result of these experiments for a sample set of parameters. (Graphs showing the impact of the various parameters are available in [7].) Experiment E5 studies the effectiveness of the procedure from Section 5.2.3 for choosing the sizes of the chunks we deploy (see Fig. 5); and finally Experiment E6 studies the impact of varying the ratio of potential replication (see Fig. 6).

A. Experiment E1: varying workload size. When $W_{(\text{ttl})} = 1$, the opportunities for replicating work are maximum. As one would expect in this case, $\Sigma_{\text{random-rep}}$ often dominates $\Sigma_{\text{no-rep}}$. Replication is therefore worth considering—but it must be implemented in a meaningful way: Σ_{brute} almost always achieves very poor performance. Another obvious conclusion is that when there is very little room for replication, i.e., when $W_{(\text{ttl})}$ is close to p, $\Sigma_{\text{no-rep}}$, $\Sigma_{\text{cyclic-rep}}$, and Σ_{greedy} achieve quite similar performance. In all cases, $\Sigma_{\text{cyclic-rep}}$ achieves better performance than $\Sigma_{\text{no-rep}}$. This is significant when $W_{(\text{ttl})}$ is small relative to pX. On every instance, Σ_{greedy} exhibits the best performance.

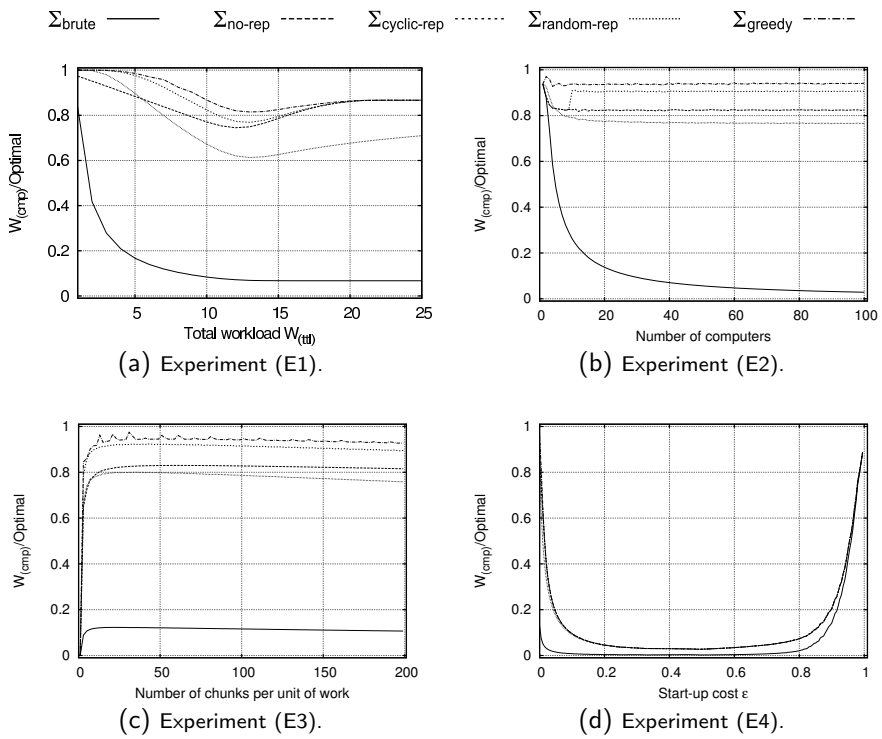
B. Experiment E2: varying number of computers. The performance of our heuristics is generally not impacted when the number of computers p grows while the overall work-to-computer ratio $W_{(\mathrm{ttl})}/p$ is kept constant. (The exception is Σ_{brute} whose performance drops dramatically.) As the ratio $W_{(\mathrm{ttl})}/p$ increases, there is less opportunity for replication, so efficient use of resources becomes more complicated; this leads to overall worse performance by all heuristics.

C. Experiment E3: varying number of chunks. When the start-up cost ε is very small, one is always better off deploying work in a larger number of chunks (i.e., doing more checkpointing), because having small chunks reduces the loss of work in progress when a computer is interrupted. However, for larger values of ε, one must be more cautious in choosing chunk sizes, because the "penalty" for each additional chunk/checkpoint then negatively impacts the expected work production; when ε is large, this impact is dramatic. It is not clear that the loss of expected work would be significant in an intermediate case such as $\varepsilon = 0.001$. But even in this case, the performance starts to decrease as the number of chunks increases. Of course, one must exercise special care in choosing the exact number of chunks when scheduling using $\Sigma_{\mathsf{cyclic\text{-}rep}}$, for that heuristic's performance fluctuates depending on whether the number of computers is relatively prime to the number of chunks. As a positive sidenote from this experiment, the general shapes of the performance curves corroborate the unimodal assumption proposed at the end of Section 5.2.3. Note that because the studied parameter is not *the overall number of chunks* but, rather, *the number of chunks per unit of work*, the number of computers has no significant impact on performance (except, obviously, for $\Sigma_{\mathsf{cyclic\text{-}rep}}$).

D. Experiment E4: varying the start-up cost ε. One observes a dramatic drop in performance as ε is allowed to grow. Indeed, when ε is large, e.g., when $\varepsilon \geq 0.05$ (roughly), very few chunks can be executed on a computer before it is interrupted. In these configurations, performance depends mainly on the size of chunks relative to the interruption times in the instance. There is no way to design good heuristics on average (compared to $\Sigma_{\mathsf{omniscient}}$) and all heuristics have poor average performance. This effect gets even worse with larger numbers of chunks per unit of work. Indeed, as ε approaches 1, the proportion of cases where even $\Sigma_{\mathsf{omniscient}}$ does not complete any work increases. (Recall: In pathological cases, all heuristics have performance ratio 1.)

Due to the poor performance of $\Sigma_{\mathsf{random\text{-}rep}}$, we do not consider this heuristic in the following experiments.

E. Experiment E5: automatic inference of chunk size. This experiment replicates experiment E1 except that, for each instance and each heuristic, the size of chunks is no longer given but is, rather, inferred algorithmically using the procedure of Section 5.2.3. In Fig. 5, for each heuristic, we plot the average performance when considering only the x% best instances for that heuristic. The performance for 100% is thus the average performance over all 760,000 instances. One observes Σ_{greedy} achieving at least 85.2% of the optimal work production of $\Sigma_{\mathsf{omniscient}}$, with $\Sigma_{\mathsf{no\text{-}rep}}$ close behind with 79.6% of the optimal. Thus, on average, Σ_{greedy} is 27.4% closer to optimal than is $\Sigma_{\mathsf{no\text{-}rep}}$. Further-



(a) Experiment (E1). (b) Experiment (E2).

(c) Experiment (E3). (d) Experiment (E4).

Fig. 4 Experiments (E1) through (E4) with 25 computers, 147 chunks, $\varepsilon = 0.0010$, and $W_{(\mathrm{ttl})} = 0.3p$.

more, in more than 21% of the instances, Σ_{greedy} is almost optimal, achieving more than 99.5% of the optimal work production. $\Sigma_{\mathsf{cyclic\text{-}rep}}$'s work production is quite close to Σ_{greedy}'s.

F. Experiment E6: varying ratio of potential replication. This experiment is dedicated to assessing the impact of the ratio of potential replication $pX/W_{(\mathrm{ttl})}$. We fixed the overall workload to $W_{(\mathrm{ttl})} = 10$ units and allowed p to assume all integral values between 10 and 100 (with the earlier mentioned four choices for the value of ε). We considered 1000 random instances of each set of parameters. The results are presented in Fig. 6. $\Sigma_{\mathsf{cyclic\text{-}rep}}$ and Σ_{greedy} always have better performance than $\Sigma_{\mathsf{no\text{-}rep}}$, and the difference is very significant as soon as the ratio of potential replication exceeds 2. Overall, Σ_{greedy} has better and more regular performance than $\Sigma_{\mathsf{cyclic\text{-}rep}}$. In contrast with Σ_{greedy}, $\Sigma_{\mathsf{cyclic\text{-}rep}}$ takes almost no advantage of the possibility of replication when the potential for replication is smaller than 2.

Summarizing the idealized experiments. The experiments show that careful use of work replication can significantly improve the performance of heuristics. The greedy heuristic Σ_{greedy} always delivers good performance, it is never outperformed by any other heuristic—on average it delivers the best performance

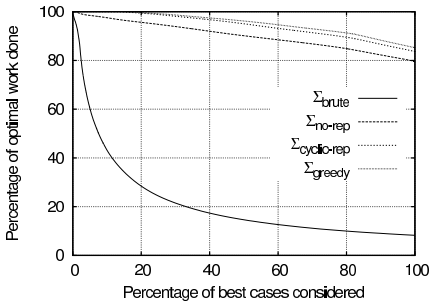


Fig. 5 Experiment (E5): performance with automatic inference of chunk sizes.

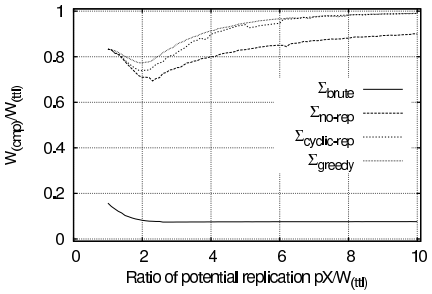


Fig. 6 Experiment (E6): impact of the ratio of potential replication, $pX/W_{(\mathrm{ttl})}$.

on every configuration—and in favorable cases, it performs significantly better than any other heuristic.

7 Experiments Based on Actual Traces

Most of the results in this paper focus on the linear risk model. Three notable exceptions that are relevant to the current study are the following results that relate to scheduling for arbitrary nondecreasing risk functions: (1) Theorem 5, which supplies guidelines for crafting optimal schedules for two workers under the free-initiation model; (2) Theorem 9, which exposes a close relationship between the expected work production under the free- and the charged-initiation models; (3) Theorem 6 which describes an asymptotically optimal schedule for two workers under the free-initiation model. Inspired by the cited theoretical results and by the development in Section 5.2.2, we devote Section 7.1 to developing group-scheduling heuristics that are intended for use with arbitrary risk functions. We then evaluate these heuristics using actual traces, in Section 7.2.

7.1 Heuristics for Arbitrary Interruption Risk

Based on our results to this point, it is natural to focus only on schedules that deploy equal-size chunks when scheduling for the general setting of p workers under arbitrary nondecreasing risk functions. First, we know that such equal-size-chunk schedules are asymptotically optimal when scheduling one or two workers (Theorem 3 and Theorem 6). Second, the group-scheduling heuristics of Section 5.2 deploy work in equal-size chunks, and this class of schedules are structurally quite attractive as we contemplate how to deploy work on arbitrary numbers of computers.

Elaborating on the second point, we note that, with the sole exception of the greedy scheduling heuristic, the underlying risk function does not play any role in the definition of any of our group-scheduling heuristics. The underlying risk function *does* have an impact on the choice of the optimal number of

chunks (see Section 5.2.3) but that is the function's only impact. Therefore, adapting any of our group-scheduling heuristics (other than the greedy heuristic) to another risk function requires only changing the number of chunks that the heuristic works with.

The preceding is both good and bad news: it is good news since adapting almost all heuristics is easy; it is bad news because the sole exception is the heuristic that dominated in all of our tests. Therefore, it is worthwhile working a bit to adapt the greedy scheduling heuristic for arbitrary interruption risk. In fact, this is not so hard: after k steps, the probability that a chunk in group j will be interrupted is proportional to the product $\prod_{i=1}^{k} Pr(\widehat{C}_{ij}^{(\mathsf{greedy})} sl/n)$. We can, therefore, adapt the greedy scheduling heuristic by using this general expression for the probability, in place of the expression specialized for the linear risk model, i.e., $\prod_{i=1}^{k} \widehat{C}_{ij}^{(\mathsf{greedy})}$.

We are now ready to assess the quality of our heuristics, using actual activity traces.

7.2 Trace-based Experiments

7.2.1 Traces and Methodology

We evaluate our scheduling heuristics using eight traces that recorded, for each computer in an assemblage, the lengths of the different time intervals during which the computer was available. These traces are: (0) the *SDSC trace* [18, p. 33] with 5678 availability durations; (1) the *UCB trace* [2] with 19276 availability durations; (2) the *XtremWeb trace* [18, p. 33] with 8756 availability durations; (3) the *Cetus trace* [25] with 1898 availability durations; (4) the *LONG trace* [25] with 10958 availability durations; (5) the *Princeton trace* [25] with 79 availability durations; (6) the *Condor trace* [23] with 1125 availability durations; and (7) the *CSIL trace* [23] with 927 availability durations.

We normalized traces so that the longest availability interval for each was exactly 1 (so that we could compare and average statistics over different traces). Then, from each trace, *trace*, we built a risk function, $Pr_{trace}(t)$, that specifies the probability of a computer's being interrupted by time t:

$$Pr_{trace}(t) = \frac{\text{Number of intervals in } trace \text{ shorter than } t}{\text{Number of intervals in } trace}.$$

We generated interruption instances by uniformly and randomly picking availability interval lengths in the studied trace. Therefore, we assumed implicitly that when making a scheduling decision, we only considered computers that just became available.

7.2.2 Simulation Results

A. Parameter settings. We ran the heuristics with parameter λ (see Section 5.1) set to 1.00, parameters p and ε set as in Section 6.1, and parameter $W_{(\mathrm{ttl})}$

taking all integral values in the range $[1..p]$. For each set of parameters and each trace we generated 1000 interruption scenarios.

B. Results. The aggregated simulation results are presented in Fig. 7 and Table 5. Overall, and under each studied scenario, Σ_{greedy} achieves far better results than $\Sigma_{\text{cyclic-rep}}$ and $\Sigma_{\text{no-rep}}$. The difference between Σ_{greedy} and the other heuristics increases in importance as the number of computers increases or as the size of the start-up cost decreases. The lesson: the more opportunity for work replication, the more obvious the advantage of Σ_{greedy}.

Fig. 8 presents aggregated result for each of the eight traces, and Fig. 9 presents the risk function of each of the eight traces. Trace 5 is the most similar and Trace 1 the least similar to the linear risk model. For both traces, and for all the intermediate cases, Σ_{greedy} achieves the best overall performance by a significant margin. This is the case when a lot of work can be successfully processed (Traces 0, 3, 5, and 7, for which the performance of Σ_{brute} is below 20%). This is also the case when very little work can be performed before all processors are interrupted (Traces 1 and 2).

The performance of $\Sigma_{\text{cyclic-rep}}$ is close to that of $\Sigma_{\text{no-rep}}$. On average, Σ_{greedy} closes 37% of the gap between $\Sigma_{\text{no-rep}}$ and the (omniscient) optimal heuristic.

The comparison of $\Sigma_{\text{omniscient}}$ and Σ_{greedy} shows that the latter has very good absolute performance. This proves the efficiency of static heuristics in this context.

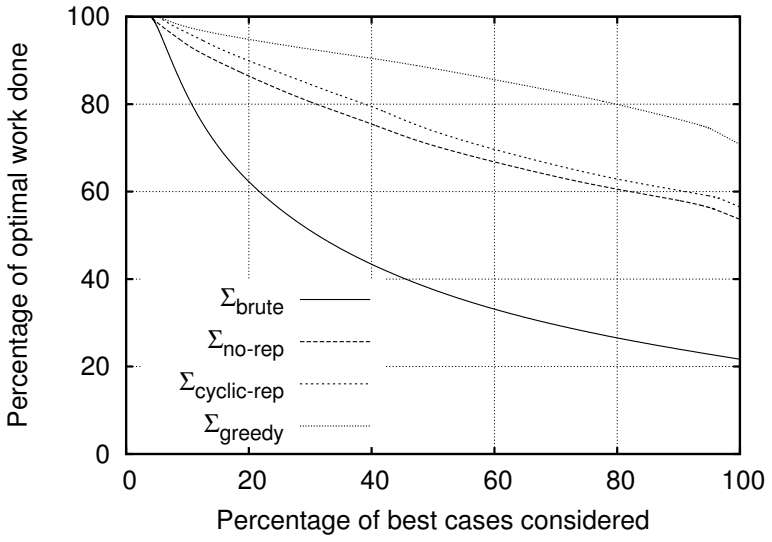
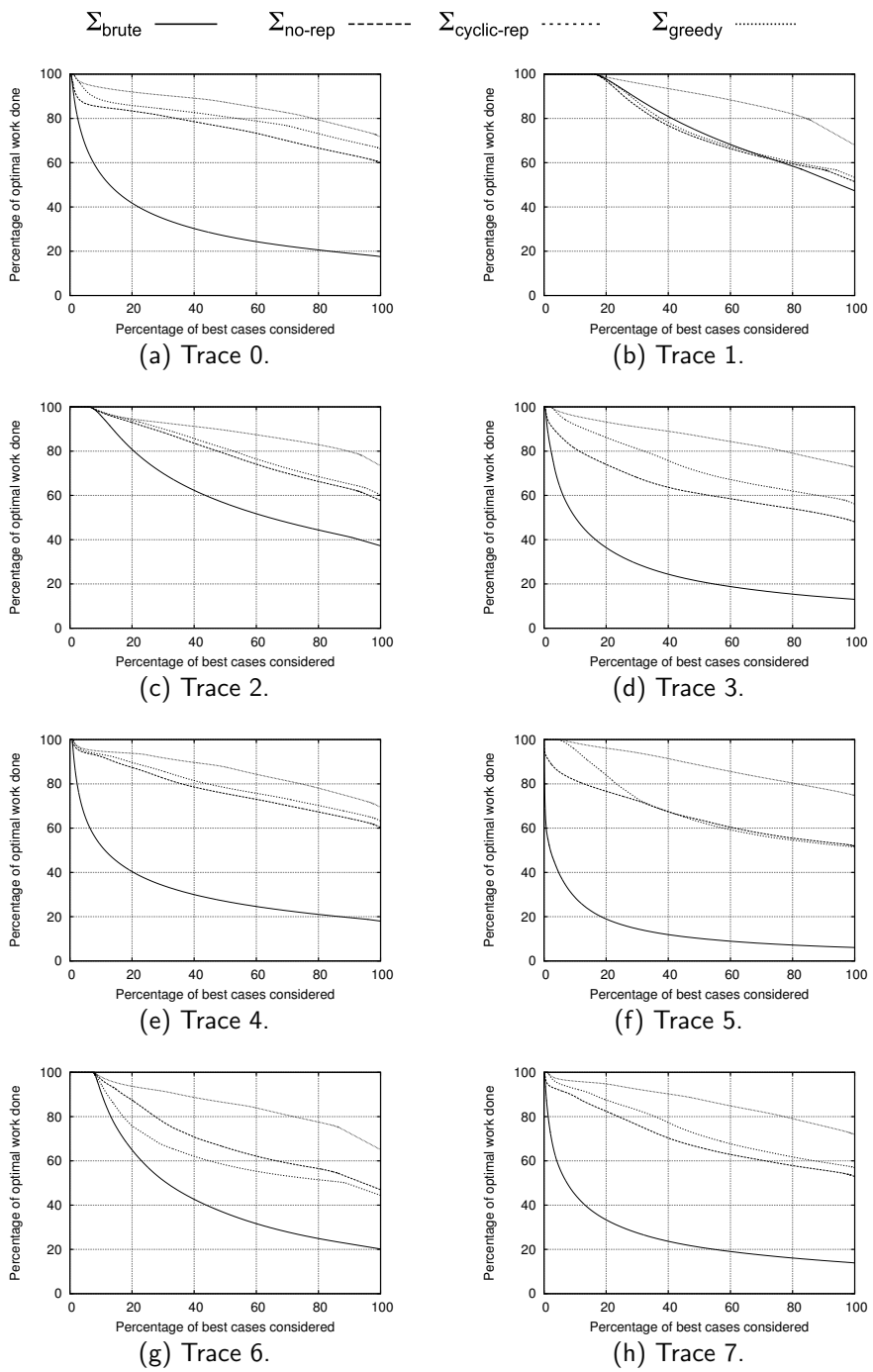


Fig. 7 Performance of the heuristics with risk functions defined by computer-availability traces. For each of the four heuristics, the curve $y = f(x)$ shows the average performance y of the heuristics when only considering the x% cases most favorable to that heuristic.



(a) Trace 0. (b) Trace 1.

(c) Trace 2. (d) Trace 3.

(e) Trace 4. (f) Trace 5.

(g) Trace 6. (h) Trace 7.

Fig. 8 Experiments with the eight different traces.

Heuristic	average	stdv
Σ_{brute}	21.7	24.2
$\Sigma_{\text{no-rep}}$	53.6	22.1
$\Sigma_{\text{cyclic-rep}}$	56.5	22.2
Σ_{greedy}	70.8	23.2

Table 5 Aggregate performance over all 6,080,000 instances for risk functions defined by computer-availability traces.

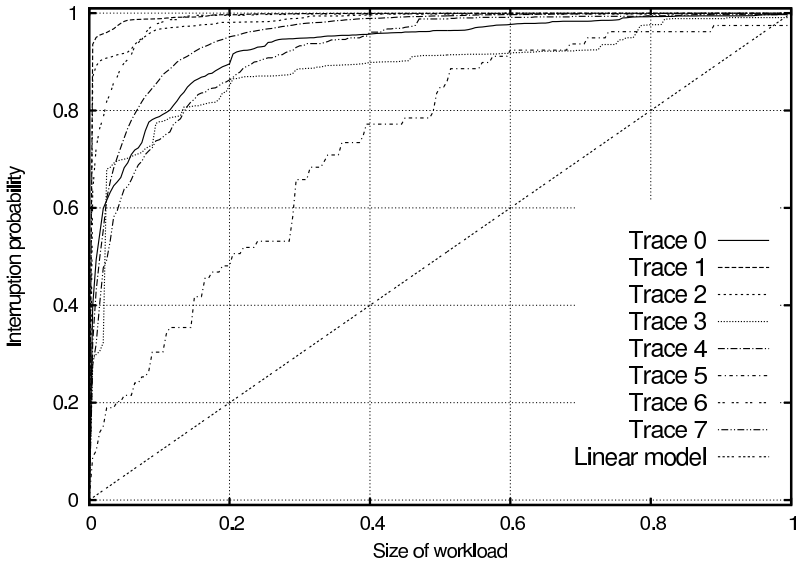


Fig. 9 Interruption probability for each of the eight traces.

8 Conclusion

We have presented a model for studying the problem of scheduling large divisible workloads on identical computers that are vulnerable (with the same risk function) to unrecoverable interruptions. Our goal has been to find schedules for allocating work to the computers and for scheduling the checkpointing of that work, in a manner that maximizes the expected aggregate amount of work completed by the computers. Most of the results we report assume that the risk of a computer's being interrupted increases *linearly* with the amount of time that the computer has been available to us; a few results provide scheduling guidelines for more general risk functions.

We have completely solved this scheduling problem for the case of $p = 1$ worker (Section 3), by deriving schedules whose expected work completion is exactly maximum, both for the free-initiation model, wherein checkpointing incurs no overhead, and the charged-initiation model, wherein checkpointing does incur an overhead. Our major theoretical focus has been the case of $p = 2$ workers (Section 4). We provide there schedules for this case whose expected work completion is either exactly or asymptotically optimal when the size of

the workload grows without bound; we also provide guidelines for deriving exactly optimal schedules. The complexity of the development in Section 4 suggests that the general case of p workers will be prohibitively difficult, even for deriving *asymptotically* optimal schedules. Therefore, we turn in this general case to deriving a number of well-structured heuristics whose quality can be assessed via explicit expressions for their expected work outputs (Section 5). Numerical evaluations suggest that one of our six group heuristics is the winner in terms of performance. An extensive suite of simulation experiments suggests that the "winner" in the competition of Section 5 provides schedules with good expected work output, and that it dominates the reference heuristics (Section 6). Finally, building on the insight gained studying the linear risk model, we turned our attention to general risk models (Section 7). We adapted our p-computer heuristics to general risk functions. Extensive simulations using actual traces of computer availabilities suggest that the winner of Sections 5 and 6 also dominates reference heuristics in the presence of general risk functions and is a very efficient heuristic. Furthermore, our simulations show that static heuristics have overall very good absolute performance.

Much remains to be done regarding this important problem area, but two directions stand out as perhaps the major outstanding challenges. One of these is to extend our study to include heterogeneous assemblages of workers, whose constituent computers differ in speed and other computational resources. We have already embarked on this study, with an initial report in [8]. Another great challenge, when the assemblages are heterogeneous, but even when they are homogeneous, is to allow the assemblage's computers to be subject to differing probabilities of being interrupted.

Acknowledgments. The work of A. Benoit, Y. Robert and F. Vivien was supported in part by the ANR StochaGrid and RESCUE projects. The work of A. Rosenberg was supported in part by US NSF Grants CNS-0842578 and CNS-0905399.

The authors thank Joshua Wingstrom for access to the availability traces.

References

1. Anglano, C., Brevik, J., Canonico, M., Nurmi, D., Wolski, R.: Fault-aware scheduling for bag-of-tasks applications on desktop grids. In: GRID '06, pp. 56–63. IEEE Computer Society (2006)
2. Arpaci, R.H., Dusseau, A.C., Vahdat, A.M., Liu, L.T., Anderson, T.E., Patterson, D.A.: The interaction of parallel and sequential workloads on a network of workstations. In: SIGMETRICS '95/PERFORMANCE '95, pp. 267–278. ACM (1995)
3. Awerbuch, B., Azar, Y., Fiat, A., Leighton, F.T.: Making commitments in the face of uncertainty: How to pick a winner almost every time. In: 28th ACM SToC, pp. 519–530 (1996)
4. Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling divisible loads on star and tree networks: results and open problems. IEEE TPDS **16**(3), 207–218 (2005)
5. Bender, M.A., Phillips, C.A.: Scheduling dags on asynchronous processors. In: 19th ACM SPAA, pp. 35–45 (2007)

6. Benoit, A., Robert, Y., Rosenberg, A.L., Vivien, F.: Static strategies for worksharing with unrecoverable interruptions. In: 23rd Intl. Parallel and Distributed Processing Symp. (IPDPS). IEEE Computer Society Press (2009)
7. Benoit, A., Robert, Y., Rosenberg, A.L., Vivien, F.: Static strategies for worksharing with unrecoverable interruptions. Research report RR2010-18, LIP, ENS Lyon, France (2010). `http://graal.ens-lyon.fr/~yrobert/onlinepapers/RRLIP2010-18.pdf`
8. Benoit, A., Robert, Y., Rosenberg, A.L., Vivien, F.: Static worksharing strategies for heterogeneous computers with unrecoverable interruptions. Parallel Computing (2010). To appear
9. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling Divisible Loads in Parallel and Distributed Systems. Wiley-IEEE Computer Society Press (1996)
10. Bhatt, S.N., Chung, F.R.K., Leighton, F.T., Rosenberg, A.L.: An optimal strategies for cycle-stealing in networks of workstations. IEEE Trans. Computers **46**(5), 545–557 (1997)
11. Buyya, R., Abramson, D., Giddy, J.: A case for economy grid architecture for service-oriented grid computing. In: 10th Heterogeneous Computing Workshop. IEEE Computer Society (2001)
12. Cirne, W., Marzullo, K.: The computational co-op: Gathering clusters into a metacomputer. In: 13th Intl. Parallel Processing Symp. (IPPS), pp. 160–166 (1999)
13. Foster, I., Kesselman, C. (eds.): The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (2004)
14. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. Intl. J. High Performance Computing Applications **15**(3), 200–222 (2001)
15. Gallet, M., Robert, Y., Vivien, F.: Comments on "design and performance evaluation of load distribution strategies for multiple loads on heterogeneous linear daisy chain networks". J. Parallel Distributed Computing **68**(7), 1021–1031 (2008)
16. Gallet, M., Robert, Y., Vivien, F.: Divisible load scheduling. In: Introduction to Scheduling. Chapman and Hall/CRC Press (2009)
17. Gao, L., Malewicz, G.: Toward maximizing the quality of results of dependent tasks computed unreliably. Theory Comput. Syst. **41**(4), 731–752 (2007)
18. Kondo, D.: Scheduling task parallel applications for rapid turnaround on enterprise desktop grids. Ph.D. thesis, University of California at San Diego (2005)
19. Kondo, D., Casanova, H., Wing, E., Berman, F.: Models and scheduling mechanisms for global computing applications. In: 16th Intl. Parallel and Distr. Processing Symp. (IPDPS) (2002)
20. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Leboisky, M.: SETI@home - Massively Distributed Computing for SETI. Computing in Science & Engineering **3**(1), 78–83 (2001)
21. Litzkow, M.J., Livny, M., Mutka, M.W.: Condor - a hunter of idle workstations. In: ICDCS, pp. 104–111 (1988)
22. Malewicz, G., Rosenberg, A.L., Yurkewych, M.: Toward a theory for scheduling dags in internet-based computing. IEEE Trans. Computers **55**(6), 757–768 (2006)
23. Nurmi, D., Brevik, J., Wolski, R.: Modeling machine availability in enterprise and wide-area distributed computing environments. In: Euro-Par 2005, vol. LNCS 3648, pp. 432–441 (2005)
24. Pfister, G.F.: In Search of Clusters. Prentice-Hall (1995)
25. Plank, J., Elwasif, W.: Experimental assessment of workstation failures and their impact on checkpointing systems. In: Fault-Tolerant Computing, 1998, pp. 48–57 (1998). DOI 10.1109/FTCS.1998.689454
26. Rosenberg, A.L.: Guidelines for data-parallel cycle-stealing in networks of workstations i: On maximizing expected output. J. Parallel Distrib. Comput. **59**(1), 31–53 (1999)
27. Rosenberg, A.L.: Guidelines for data-parallel cycle-stealing in networks of workstations ii: On maximizing guaranteed output. Intl. J. Foundations of Computer Science **11**(1), 183–204 (2000)
28. Rosenberg, A.L.: Optimal schedules for cycle-stealing in a network of workstations with a bag-of-tasks workload. IEEE Trans. Parallel Distrib. Syst. **13**(2), 179–191 (2002)
29. Rosenberg, A.L.: Changing challenges for collaborative algorithmics. In: A. Zomaya (ed.) Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies, pp. 1–44. Springer (2006)

30. White, S., Torney, D.: Use of a workstation cluster for the physical mapping of chromosomes. SIAM NEWS pp. 14–17 (1993)
31. Wingstrom, J., Casanova, H.: Probabilistic allocation of tasks on desktop grids. In: Proceedings of PCGrid. IEEE CS Press (2008)